Parallele Algorithmen zur Lösung des Capacitated-Vehicle-Routing-Problems

Evaluierung des Einsatzes von Grafikkarten

Vom Fachbereich Wirtschaftswissenschaften
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doctor rerum politicarum (Dr. rer. pol.)
genehmigte

D i s s e r t a t i o n

vorgelegt von

Dipl.-Kfm. techn. Bastian Sand

Tag der mündlichen Prüfung:	17. Juli 2013
Dekan:	Prof. Dr. Stefan Roth
Vorsitzender:	Prof. Dr. Jan Wenzelburger
Berichterstatter:	1. Prof. Dr. Oliver Wendt
	2. Prof. Dr. Reinhold Hölscher

D 386
(2013)

Für meine Eltern

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1C	1-control
CO	Collegial
COSA	Cooperative-Simulated-Annealing
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CVRP	Capacitated-Vehicle-Routing-Problem
DCVRP	Distance-Constrained-Capacitated-Vehicle-Routing-Problem
Flop/s	Floating Point Operations per Second
GA	Genetischer Algorithmus
GPGPU	General-Purpose Computation on Graphics Processing Unit
GPU	Graphics Processing Unit
HPC	High-Performance Computing
KC	Knowledge Collegial
KS	Knowledge Synchronization
MA	Memetischer Algorithmus
MDVRP	Multi-Depot-Vehicle-Routing-Problem
MIMD	Multiple-Instruction Stream-Multiple-Data Stream
MISD	Multiple-Instruction Stream-Single-Data Stream Organization
MPDS	Multiple Initial Points/Populations, Different Search Strategies
MPSS	Multiple Initial Points/Populations, Same Search Strategies
OX	Order Crossover
pC	p-control
PVRP	Periodic-Vehicle-Routing-Problems
RS	Rigid Synchronization
SCaC	Search Control and Communications
SCC	Search Control Cardinality
SD	Search Differentiation
SimA	Simulated Annealing

SIMD	Single-Instruction Stream-Multiple-Data Stream
SISD	Single-Instruction Stream-Single-Data Stream Organization
SM	Streaming Multiprocessors
SMem	Shared Memory
SP	Streaming Processor
SPDS	Same Initial Point/Population, Different Search Strategies
SPMD	Single-Program Multiple-Data
SPSS	Same Initial Point/Population, Same Search Strategies
TS	Tabusuche
TSP	Traveling-Salesman-Problem
VNS	Variable Neighborhood Search
VRP	Vehicle-Routing-Problem
VRPB	Vehicle-Routing-Problem with Backhauls
VRPBTW	Vehicle-Routing-Problem with Backhauls and Time Windows
VRPPD	Vehicle-Routing-Problem with Pickup and Delivery
VRPPDTW	Vehicle-Routing-Problem with Pickup and Deliveries and Time Windows
VRPTW	Vehicle-Routing-Problem with Time Windows

Symbolverzeichnis

A	Kantenmenge eines CVRPs
C	Maximalkapazität eines Fahrzeugs
c_Δ	Betrag der Änderung des Zielfunktionswertes von der aktuellen Lösung zu einer neuen Lösung
$C_{r_{v_i}}$	Kapazitätsbedarf der Route r_{v_i}
c_{v_i,v_j}	Fahrtkosten, die für die Strecke von Knoten v_i zu v_j anfallen
C_p	Kosten der Parallelisierung
d_{v_i}	Nachfrage von Knoten v_i
$\overleftarrow{d}_{v_i}$	Rückwärtskapazität von Knoten v_i
$\overrightarrow{d}_{v_i}$	Vorwärtskapazität von Knoten v_i
$E_p(n)$	Effizienz
$f(s)$	Zielfunktion
$H_p(n)$	Overhead
i_l	Länge der Knotensequenz, die aus Route r_{v_i} entfernt wird
j_l	Länge der Knotensequenz + 1, die aus Route r_{v_j} entfernt wird
k_l	Fahrzeug
m	Anzahl der Fahrzeuge
M	Move bzw. Operator
n	Anzahl der Kunden eines CVRPs
$N(s)$	Menge der Lösungen, die in Nachbarschaft zur Lösung s liegen
$N_M(s)$	Menge der Lösungen, die in Nachbarschaft zur Lösung s liegen und durch Move M resultieren
p	Anzahl der Prozessoren
P_A	Annahmewahrscheinlichkeit
r_{v_i}	Route bzw. Fahrzeug, das Knoten v_i bedient
s	Lösung aus der Lösungsmenge S
S	Lösungsmenge
$S_p(n)$	Speedup
T	Temperatur
$T^*(n)$	Laufzeit der besten sequentiellen Implementierung eines Algorithmus

$T_p(n)$	Laufzeit eines parallelen Algorithmus
TLL	Länge der Tabuliste
V	Knotenmenge eines CVRPs
v_0, v_{n+1}	Depotknoten
v_i	Knoten in einem CVRP

1 Einleitung

1.1 Untersuchungsgegenstand

Die Tourenplanung ist ein weit erforschtes Themenspektrum des Operations Research. Ein typischer Vertreter ist das Vehicle-Routing-Problem (VRP), das zum Ziel hat, eine Menge von Kunden von einem oder mehreren Depots aus mit einer Fahrzeugflotte (homogen oder heterogen) zu beliefern und zuvor definierte Zielkriterien, wie bspw. die zurückgelegte Distanz der eingesetzten Fahrzeuge oder Kundenzufriedenheit, zu optimieren. Das Grundproblem wird in der Literatur um viele Restriktionen erweitert, die im Folgenden unter dem VRP subsumiert werden. Die in der Arbeit betrachtete Variante, das Capacitated-Vehicle-Routing-Problem (CVRP), enthält lediglich eine Kapazitätsbeschränkung der Fahrzeuge. Das heißt, sobald die Kapazitätsschranke erreicht ist, dürfen keine weiteren Kunden mit dem Fahrzeug bedient werden. Bei dem CVRP handelt es sich um eine Abstraktion des Postproblems. Das rührt daher, dass von einem Depot eines Postunternehmens aus täglich eine Vielzahl von Paketen ausgeliefert werden muss. Damit wird die praktische Relevanz des CVRPs deutlich.

Um die Touren der Fahrzeuge möglichst kostengünstig zu planen, existieren viele verschiedene Verfahren. Bei den Lösungsverfahren kann man grob zwischen exakten, heuristischen und metaheuristischen Ansätzen differenzieren. Exakte Methoden garantieren als Ergebnis die optimale Lösung eines Problems. Gleichzeitig benötigen sie jedoch zur Bestimmung des optimalen Tourenplans sehr viel Rechenzeit, da bereits das CVRP NP-schwer ist (Toth und Vigo, 2002b, S. 8). Exakte Lösungsverfahren kommen demnach nur für Probleminstanzen mit einer geringen Anzahl von Kunden in Betracht. Um für Probleme mit vielen Kunden eine gültige und gleichzeitig zufriedenstellende[1] Lösung zu finden, werden Heuristiken eingesetzt. Sie ermöglichen es, in relativ kurzer Rechenzeit eine gute Lösung, aber nicht notwendigerweise die Optimallösung für ein VRP, zu finden. Ebenfalls zu dieser Kategorie sind Metaheuristiken zu zählen, die jedoch, im Vergleich zu einer problemspezifischen Heuristik, problemunabhängig sind, d.h. sie sind auf unterschiedlichste Problemstellungen in gleicher Art und Weise anwendbar.

Die Lösungsverfahren aller Kategorien werden auf vielen verschiedenen Hardwareplattformen ausgeführt, wobei ein Großteil handelsübliche Central Processing Units (CPU) verwendet. In jüngerer Vergangenheit wird auch die zunehmende Anzahl von Kernen aktueller CPU-Architekturen genutzt (vgl. Jin et al. (2012)). Neben den Lösungsverfahren, die nur auf einem Rechner ausgeführt werden, finden sich andere, die von verteilten Computersystemen Gebrauch machen (vgl. bspw. Groër et al. (2010)). In diesem Fall ist es möglich, dass die Rechner, die in den Lösungsprozess eingebunden sind, unterschiedliche CPUs besitzen. Wie die Aufgaben letztendlich auf dem zusammengeschlossenen System

[1] Zufriedenstellend ist in diesem Zusammenhang sowohl im Hinblick auf die Dauer der Berechnung als auch der Lösungsgüte, d.h. im Falle des CVRPs auf eine möglichste geringe zurücklegte Gesamtdistanz, zu sehen.

verteilt werden, hängt von der Problemstellung ebenso ab wie von dem verwendeten Algorithmus.[2] Seit etwa 2001 hat sich ein Forschungsfeld aufgetan, das sich mit allgemeinen Berechnungen auf Grafikkarten, auch General-Purpose Computation on Graphics Processing Unit (GPGPU) genannt, beschäftigt (Sanders und Kandrot, 2010, S. 5f.) und das sich damit mit dem Konzept der verteilten bzw. parallelen Berechnung auseinandersetzt. Die eigentliche Aufgabe von Graphics Processing Units (GPU) liegt in der Ausgabe eines Bildes am Monitor. Angetrieben durch einen immer größeren Detailreichtum in Computerspielen, hat sich die Rechenleistung von Grafikkarten sehr schnell gesteigert (Nickolls und Dally, 2010, S. 56), sodass die theoretisch möglichen Fließkommaoperationen pro Sekunde (Flop/s) in einfacher Genauigkeit die einer CPU um ein Vielfaches übersteigen (vgl. Abbildung 1.1). GPUs haben eine solch hohe Rechenleistung erreicht, dass sie in den schnellsten Rechnern der Welt zum Einsatz kommen (Stiller, 2010). Dieser immense Leistungsschub hat Forscher dazu bewegt, GPUs für rechenintensive Verfahren, die sich nicht nur auf die Grafikausgabe beschränken, einzusetzen.

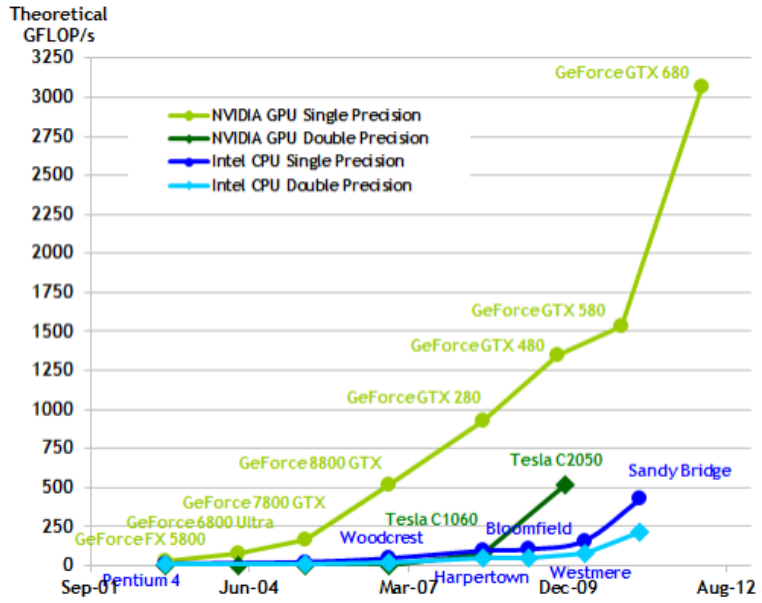


Abbildung 1.1: Entwicklung der Anzahl der Fließkommazahlberechnungen pro Sekunde im Zeitverlauf (Quelle: Nvidia (2012))

Allen verteilten Hardwareplattformen ist gemein, dass es zur vollständigen Hardwarenutzung unvermeidlich ist, parallele Algorithmen zu entwickeln. Auch wenn Heuristiken bzw. Metaheuristiken i.d.R. eine wesentlich geringere Ausführungszeit im Vergleich zu exakten Verfahren aufweisen, ist auch hier der Wunsch nach immer mehr Rechnerressourcen gegeben. Das resultiert einerseits daraus, dass man Probleme schneller lösen möchte und andererseits Lösungen für größere Probleme, die bisher

[2] Meist steht die Frage im Mittelpunkt, ob eine Dekomposition eines Problems in unabhängige Teilprobleme möglich ist. Je besser sich ein Problem in unabhängige Teilprobleme untergliedern lässt, umso besser lässt sich ein (physisch) verteiltes System zum Lösen einsetzen. Jeder eingesetzte Rechner kann unabhängig von den anderen Rechnern sein Teilproblem lösen, ohne auf die Inputdaten anderer Rechner warten zu müssen. Sobald die Aufteilung in unabhängige Teilprobleme nicht mehr oder nur bedingt möglich ist, kann bspw. der Kommunikations- und Synchronisationsaufwand schnell steigen, da die Outputdaten eines Rechners die Inputdaten eines anderen sein können. Im schlechtesten Fall ist keine Dekomposition möglich, da zur Lösung eines Problems immer der vorhergehende Schritt nötig ist. Dann spielt es keine Rolle, ob der Algorithmus nur auf einem Rechner oder von mehreren Rechnern abgearbeitet wird, da er in beiden Fällen rein sequentiell abläuft.

nicht mit den zur Verfügung stehenden Ressourcen zu bearbeiten sind, ermitteln möchte. Weiterhin ist der Einsatz von GPUs nicht nur aus dem Wunsch nach kurzen Ausführungszeiten heraus motiviert, sondern auch aus den Kosten, die für Rechenleistung aufzuwenden sind. So liefert ein Intel Core i7-965 eine Single-Precision-Leistung von 51,2 GFlop/s (Intel, 2011) und kostet ca. 850 Euro (Quelle: Preissuchmaschine von Google am 30.5.2011). Demgegenüber steht eine Nvidia Geforce 9500 GT, die eine theoretische Single-Precision-Leistung von 134,4 GFlop/s (GPUReview, o.J.) liefert, jedoch lediglich ca. 60 Euro (Quelle: Preissuchmaschine von Google am 30.5.2011) kostet.[3] Vor diesem Hintergrund sind die Untersuchungsgegenstände dieser Arbeit parallele (meta)heuristische Algorithmen, die die GPU nutzen und zur Lösung des CVRPs verwendet werden.

1.2 Untersuchungsziele

Ziel der Arbeit ist es, die Auswirkungen des Einsatzes GPU-basierter Heuristiken bzw. Metaheuristiken zum Lösen des CVRPs anhand von drei Dimensionen zu analysieren. Dazu zählen die Performance, die Lösungsgüte sowie eine wirtschaftliche Bewertung.

Im Rahmen der Performanceanalyse soll untersucht werden, inwiefern sich die Ausführungsgeschwindigkeit eines Algorithmus ändert, wenn im Lösungsprozess eine GPU zur Anwendung kommt. Basis vieler metaheuristischer Verfahren sind lokale Suchoperatoren. Deshalb findet eine Implementierung von GPU-basierten Suchoperatoren statt. Ziel ist es, Gestaltungsmöglichkeiten der Parallelisierung der Operatoren auf Grafikkarten aufzuzeigen und die Ansätze mit numerischen Analysen zu bewerten. Hierbei sollen sowohl Unterschiede zwischen den Operatoren als auch innerhalb der Operatoren durch Verwendung unterschiedlicher Speicherarten auf der GPU aufgezeigt werden. Weiterhin werden die Auswirkungen des Einsatzes einer Tabuliste, die im weiteren Verlauf der Arbeit in einer Tabusuche (TS) eingesetzt wird, näher betrachtet. In diesen Analysen wird lediglich die Performancedimension berücksichtigt. Die Gestaltungsempfehlungen resultieren aus den Laufzeiten der Operatoren bzw. aus dem Laufzeitvergleich der GPU- und CPU-Implementierung. Die Basis der lokalen Suchoperatoren bildet das Konzept von Toth und Vigo (2003). Dadurch unterscheiden sich die Untersuchungen dieser Arbeit von denen von Schulz (2013), der auf einer gänzlich anderen Idee bei der Evaluierung lokaler Suchoperatoren auf der GPU aufbaut.

Die Dimension der Lösungsgüte soll im Rahmen von Metaheuristiken betrachtet werden. Im Gegensatz zu exakten Verfahren ist die Lösungsgüte bei der Betrachtung von Metaheuristiken von fundamentaler Bedeutung für die Bewertung eines Algorithmus, wenn man bedenkt, dass durch eine hohe Lösungsqualität im Sinne einer geringen Distanz, die zur Belieferung von Kunden zurückgelegt werden muss, Kilometer und somit Kosten reduziert werden.

Die lokalen Suchoperatoren bilden die Grundlage für die weiteren Untersuchungen. Dabei werden sie zunächst in einer Metaheuristik eingesetzt und deren Performance - sowohl hinsichtlich der Lösungsgüte als auch hinsichtlich der Rechendauer - analysiert. Hierbei soll weiterhin versucht werden, die Ressourcen, die eine GPU bereitstellt, vollständig auszunutzen. Weiterhin resultieren aus diesen Untersuchungen Ergebnisse zu vielen bekannten Benchmarks der Literatur. Bisher veröffentlicht kein Beitrag der Literatur, der sich mit Algorithmen zur Lösung von VRPs auf der GPU beschäftigt, konkrete Lösungen

[3] Hierbei sollte beachtet werden, dass es sich nur um die Anschaffungskosten handelt. Die Betriebskosten werden nicht berücksichtigt.

zu bekannten Benchmarks, sondern lediglich Gestaltungsempfehlungen (vgl. z.B. Schulz (2013)). Damit ist es möglich, eine erste Einordnung von GPU-basierten Metaheuristiken zur Lösung von CVRPs mit State-of-the-Art-Algorithmen der Literatur vorzunehmen. Weiterhin soll damit gezeigt werden, wie sich durch Hinzunahme einer GPU die Qualität der Lösungen verändert.

Die wirtschaftliche Bewertung der Algorithmen soll anhand einer GPU-Implementierung im Vergleich zu einer CPU-Implementierung vorgenommen werden. Ziel ist es, die Auswirkungen der Hinzunahme einer GPU in den Berechnungen aufzuzeigen. Grundlage hierfür bilden die Rechencluster von Amazon.com.

1.3 Aufbau

Die Arbeit ist dreigeteilt aufgebaut. Zunächst werden Grundlagen, die für die Arbeit von fundamentaler Bedeutung sind, erläutert. Danach wird auf die Implementierung der Software eingegangen, bevor diese mittels numerischer Analysen betrachtet wird. Die Arbeit schließt mit einer kurzen Zusammenfassung.

Die Grundlagen in Kapitel 2 beginnen zunächst mit kombinatorischen Optimierungsproblemen im Allgemeinen und ihrer Bedeutung für die Betriebswirtschaft. Danach wird das VRP vorgestellt und für das CVRP eine mathematische Formulierung angegeben. Darauf folgt eine Einführung in Algorithmen, die zur Lösung kombinatorischer Optimierungsprobleme verwendet werden. Der Fokus liegt hierbei auf Verfahren für VRPs. Außerdem werden Grundlagen zur Charakterisierung paralleler Algorithmen und zum Vergleich von sequentiellen und parallelen Verfahren in diesem Kapitel betrachtet. Auch die Möglichkeiten zum Vergleich von Metaheuristiken werden näher erläutert. Abschließend findet sich ein kurzer Literaturüberblick über bereits bestehende Lösungsverfahren für VRPs; des Weiteren wird die Hardwarearchitektur der GPU, die in dieser Arbeit verwendet wird, betrachtet.

Die Implementierung der in der Arbeit verwendeten Algorithmen ist in Kapitel 3 zu finden. Dazu wird zunächst auf die Umsetzung lokaler Suchoperatoren eingegangen. Darauf aufbauend wird der Einsatz dieser Suchoperatoren in Metaheuristiken erläutert. Zunächst erhält man eine Beschreibung der Variable Neighborhood Search (VNS) mit Bestandteilen des Simulated Annealing (SimA). Danach wird die implementierte TS beschrieben, bevor auf die Umsetzung eines genetischen Algorithmus (GA), der die zuvor erläuterten Heuristiken verwendet, eingegangen wird. Der Aufbau erfolgt dabei nach dem Schema, wie sich die Software im Zeitverlauf entwickelt hat. Das heißt, es wurde mit den lokalen Suchoperatoren begonnen. Diese wurden dann in Metaheuristiken integriert, die dann wiederum immer weiter entwickelt wurden.

In Abschnitt 4 werden die Algorithmen, deren Implementierung zuvor beschrieben wurde, analysiert. Dazu werden zunächst Tests zu lokalen Suchoperatoren und die Auswirkungen unterschiedlicher Konfigurationen, die sich durch die zugrunde liegende Hardwarearchitektur ergeben, betrachtet. Danach werden die Metaheuristiken einer genaueren Betrachtung unterzogen. Das Hauptaugenmerk liegt auf der Lösungsqualität sowie ihrer Leistung im Vergleich zu Verfahren der Literatur. Die numerischen Analysen schließen mit einer Wirtschaftlichkeitsbetrachtung einer GPU-basierten Metaheuristik gegenüber ihrem CPU-Pendant. Schließlich werden die Ergebnisse der Arbeit in Kapitel 5 zusammengefasst und ein Ausblick auf zukünftige Forschungsmöglichkeiten gegeben.

2 Grundlagen von Vehicle-Routing-Problemen, Lösungsverfahren und Parallelisierung

2.1 Kombinatorische Optimierungsprobleme

Der Hauptuntersuchungsgegenstand dieser Arbeit, das CVRP, ist ein kombinatorisches Optimierungsproblem. Deshalb wird an dieser Stelle zunächst auf die Eigenschaften bzw. Definition kombinatorischer Optimierungsprobleme und ihre Bedeutung für die Betriebswirtschaft eingegangen. Nach Domschke und Scholl (2005, S. 79) kann man kombinatorische Optimierungsprobleme unterteilen in Reihenfolge-, Gruppierungs-, Zuordnungs- und Auswahlprobleme. Tourenplanungsprobleme lassen sich den Klassen der Gruppierungs- und Reihenfolgeprobleme zuordnen.

Die Lösung eines kombinatorischen Optimierungsproblems setzt sich i.d.R. aus einem Vektor $s = (s_1, ..., s_n)$ zusammen, der n Entscheidungsvariablen enthält. Wenn den Entscheidungsvariablen Werte zugeordnet sind, so spricht man von s als Lösung eines Problems. Alle Lösungen eines Optimierungsproblems sind in der Lösungsmenge S enthalten (Rothlauf, 2011, S. 13). Nach Rothlauf (2011, S. 14) sind die Entscheidungsvariablen „from a finite, discrete set“. Zur Definition einer Probleminstanz soll sich an die Formalisierung von Rothlauf (2011, S. 14) angelehnt werden. Er definiert ein Paar (S, f), wobei für alle $s \in S$ gilt, dass sie gültige Lösungen repräsentieren. Weiterhin existiert eine Funktion $f : S \to \mathbb{R}$. Es handelt sich hierbei um eine Evaluierungsfunktion aller Lösungen $s \in S$, die auch als Zielfunktion bezeichnet wird. Jeder Lösung wird eine Zahl zugeordnet, um die Lösungsgüte zu bestimmen. Mittels der Evaluierungsfunktion kann zwischen einem Maximierungs- und Minimierungsproblem unterschieden werden. Handelt es sich um ein Maximierungsproblem, so sucht man $s^* \in S$ für das gilt:

$$f(s^*) \geq f(s), \forall s \in S \tag{2.1}$$

Andernfalls sucht man $s^* \in S$ für das gilt:

$$f(s^*) \leq f(s), \forall s \in S \tag{2.2}$$

Nach Rothlauf (2011, S. 14) ist ein Optimierungsproblem „a set I of instances of a problem. A problem instance is a concrete realization of an optimization problem and an optimization problem can be viewed as a collection of problem instances with the same properties and which are generated in a similar way“.

Vielen kombinatorischen Optimierungsproblemen gemein ist, dass mit wachsender Größe der Probleminstanz die Anzahl der in Frage kommenden Lösungen exponentiell ansteigt. Dadurch steigt auch die Rechenzeit der Lösungsverfahren exponentiell an, und es existieren i.d.R. keine effizienten Verfahren zur Lösung der Probleme (Domschke und Scholl, 2005, S. 79). Ein Verfahren gilt im Rahmen der Komplexitätstheorie als effizient, „wenn seine Rechenzeit bzw. sein Rechenaufwand durch ein von der

Problemgröße [...] abhängiges Polynom nach oben beschränkt wird. Bei nichteffizienten Verfahren wächst die Rechenzeit ebenso wie der Lösungsraum mit zunehmender Problemgröße exponentiell. Probleme für die ein effizientes Lösungsverfahren bekannt ist, werden als polynomial lösbar und solche für die dies nicht der Fall ist, als NP-schwer bezeichnet“ (Domschke und Scholl, 2005, S. 79).

Die betriebswirtschaftliche Relevanz von kombinatorischen Optimierungsproblemen ergibt sich aus ihren Anwendungsfeldern. So können Maschinenbelegungsprobleme, Losgrößenplanung und Fließbandabstimmung genauso kombinatorischen Optimierungsproblemen zugeordnet werden wie bspw. die Personaleinsatzplanung (Domschke und Scholl, 2005, S. 79). Das verdeutlicht, dass kombinatorische Optimierungsprobleme in der Betriebswirtschaft äußerst breit gefächert und in nahezu jedem Bereich eines Betriebes von großer Bedeutung sind.

2.2 Einordnung und Formalisierung des Capacitated-Vehicle-Routing-Problems (CVRP)

Die Tourenplanung kann als Teil der Logistik eines Unternehmens angesehen werden (Domschke und Scholl, 2005, S. 166). Dabei umfasst die „Logistik alle Tätigkeiten, die sich auf die Bereitstellung von Gütern in der richtigen Menge und Qualität, zum richtigen Zeitpunkt, am richtigen Ort, zu den dafür minimalen Kosten beziehen. [...] Man unterscheidet [...] die Teilbereiche Beschaffungs-Logistik, innerbetriebliche oder Produktions-Logistik, Absatz- oder Distributions-Logistik und Entsorgungs-Logistik. Die Logistik überlagert somit als Querschnittsfunktion die betrieblichen Funktionsbereiche Beschaffung, Produktion und Absatz“ (Domschke und Scholl, 2005, S. 135). Auch die Tourenplanung findet sich damit in vielen Bereichen eines Unternehmens wieder. Ihre Aufgabe ist es, Transportprozesse, die bei der Beschaffung oder Distribution von Gütern hervortreten, zu optimieren (Domschke und Scholl, 2005, S. 166). Der Planungshorizont der Tourenplanung kann als kurzfristig angesehen werden, da sie zumeist täglich auszuführen ist (Arnold et al., 2008, S. 137).

Im Forschungsfeld der VRPs gibt es unzählige Varianten, die die Urform des VRPs, nämlich das CVRP, anpassen oder erweitern. Der Ursprung des CVRPs, das NP-schwer ist (Toth und Vigo, 2002b, S. 8), findet sich erstmals in der Arbeit von Dantzig und Ramser (1959), die das Problem als Truck-Dispatching-Problem bezeichnen. Die Autoren definieren das Problem als eine Generalisierung des Traveling-Salesman-Problem (TSP) vor dem praktischen Hintergrund der Ölbelieferung von Tankstellen.

Ziel des CVRPs ist es, eine bestimmte Anzahl von Kunden mit einer zuvor festgelegten maximal zur Verfügung stehenden Anzahl von Fahrzeugen zu beliefern. Dabei besitzt jedes Fahrzeug eine Kapazität, die nicht überschritten werden darf. Weiterhin muss jede Tour am Depot starten und enden. In dieser Arbeit soll das CVRP als graphtheoretisches Problem in Analogie zu Toth und Vigo (2002b, S. 6) definiert werden. Gegeben sei ein vollständiger Graph $G = (V, A)$ mit $V = \{v_0, ..., v_{n+1}\}$ als die Menge der Knoten und A als Menge der Kanten. Dabei ist n die Anzahl der Kunden, die beliefert werden müssen, und v_0 bzw. v_{n+1} repräsentiert das Depot. Weiterhin besitzt jede Kante $(v_i, v_j) \in A$ einen nichtnegativen Kostensatz c_{v_i,v_j}. Man kann differenzieren zwischen asymmetrischen und symmetrischen CVRPs. Im letztgenannten Fall und der in dieser Arbeit genutzten Variante gilt, dass $\forall v_i \in V$ und $v_j \in V$ $c_{v_i,v_j} = c_{v_j,v_i}$. Jeder Kunde $v_i \in V \setminus \{v_0, v_{n+1}\}$ besitzt eine nichtnegative Nachfrage d_{v_i} und jedes Fahrzeug die gleiche Kapazität C, die nicht überschritten werden darf. Weiterhin ist der Fuhrpark auf eine bestimmte Anzahl von Fahrzeugen m, mit denen die Kunden beliefert werden, beschränkt.

Jedes Fahrzeug wird über $k_l \in \{k_1, ..., k_m\}$ identifiziert. Somit ergibt sich in Anlehnung an Fisher und Jaikumar (1981, S.110f.) und Wendt (1995, S. 28f.) folgende mathematische Formulierung des CVRPs:

Zielfunktion:

$$D = \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{l=1}^{m} c_{v_i,v_j} \cdot x_{v_i,v_j,k_l} \rightarrow min \tag{2.3}$$

Nebenbedingungen:

$$\sum_{l=1}^{m} y_{v_i,k_l} = \begin{cases} m, & i = 0 \\ 1, & i = 1, ..., n \end{cases} \tag{2.4}$$

$$\sum_{i=1}^{n} d_{v_i} \cdot y_{v_i,k_l} \leq C \qquad \forall l \in \{1, ..., m\} \tag{2.5}$$

$$\sum_{j=1}^{n} x_{v_i,v_j,k_l} = \sum_{j=1}^{n} x_{v_j,v_i,k_l} = y_{v_i,k_l} \qquad \forall i \in \{0, ..., n\}, \forall l \in \{1, ..., m\} \tag{2.6}$$

$$\sum_{i \in SUB} \sum_{j \in SUB} x_{v_i,v_j,k_l} \leq |SUB| - 1 \qquad \forall SUB \subset \{1, ..., n\}, \forall l \in \{1, ..., m\} \tag{2.7}$$

$$x_{v_i,v_j,k_l} \in \{0; 1\} \qquad \forall i, j \in \{1, ..., n\}, \forall l \in \{1, ...m\} \tag{2.8}$$

$$y_{v_i,k_l} \in \{0; 1\} \qquad \forall i \in \{1, ..., n\}, \forall l \in \{1, ..., m\} \tag{2.9}$$

Die Entscheidungsvariable x_{v_i,v_j,k_l} ist 1, wenn die Kante zwischen den Knoten v_i und v_j von Fahrzeug k_l befahren wird und andernfalls 0. Analog dazu zeigt y_{v_i,k_l}, ob Knoten v_i von Fahrzeug k_l beliefert wird, d.h. wenn y_{v_i,k_l} 0 ist, wird v_i nicht von k_l beliefert; wenn y_{v_i,k_l} 1 ist, wird v_i von Fahrzeug k_l beliefert. Gleichung 2.3 repräsentiert die Zielfunktion des CVRPs, d.h., es gilt die zurückgelegten Touren möglichst kostenminimal zu befahren. Die Nebenbedingungen 2.4 stellen sicher, dass jeder Kunde von exakt einem Fahrzeug beliefert wird. Es ist also nicht erlaubt, die nachgefragte Menge d_{v_i} eines Kunden aufzusplitten und ihn mit zwei oder mehr Lieferungen zu bedienen. Gleichungen 2.5 sorgen dafür, dass die Kapazität der einzelnen Fahrzeuge nicht die Maximalkapazität C überschreitet. Gleichungen 2.6 verlangen, dass jedes Fahrzeug, das einen Kunden anfährt, den Kunden auch wieder verlässt. Mit Gleichungen 2.7 werden Subtouren ausgeschlossen und die beiden letzten Gleichungen stellen die Ganzzahligkeitsbedingung sicher (Wendt, 1995, S. 28).

Das soeben vorgestellte CVRP bildet die Grundklasse für eine Vielzahl weiterer Problemklassen. Abbildung 2.1 zeigt eine Übersicht über in der Literatur oft verwendete Problemklassen und vermittelt einen Eindruck davon, wie breit das Forschungsfeld rund um die Tourenplanung aufgebaut ist. So existieren Problemklassen bei denen die Routenlänge (Route length) beschränkt ist. Weiterhin gibt es Instanzen in denen Pakete bei einem Kunden abgeholt und zu einem anderen gebracht werden (Mixed service). Eine andere beliebte Erweiterung der Problemklassen ist die um Zeitfensterrestriktionen (Time Windows). Zuletzt seien noch die Instanzen genannt, bei denen Waren zu Kunden geliefert und von (anderen) Kunden abgeholt werden (Backhauling).

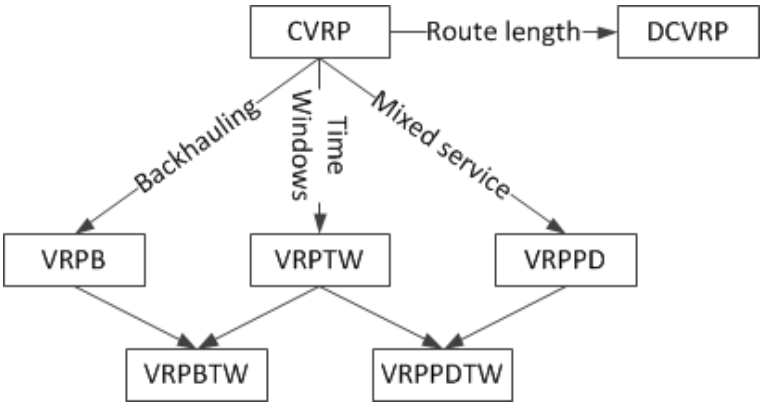


Abbildung 2.1: VRP-Klassen in Anlehnung an Toth und Vigo (2002b, S. 6)

2.3 Lösungsverfahren für kombinatorische Optimierungsprobleme

Dieser Abschnitt behandelt Verfahren zur Lösung von kombinatorischen Optimierungsproblemen, wobei der Fokus auf dem CVRP liegt. Zunächst erhält man einen Überblick über verschiedene Arten von Lösungsverfahren und deren Zusammenhänge, bevor darauf aufbauend auf Heuristiken eingegangen wird. Danach werden einige Metaheuristiken vorgestellt, wobei sich auf die beschränkt wird, die bei der Implementierung der Software in dieser Arbeit zur Anwendung kommen.

2.3.1 Einordnung von Lösungsverfahren

Die Verfahren zur Lösung des CVRP lassen sich grundsätzlich in drei Kategorien, nämlich exakte, heuristische und metaheuristische einteilen. Die exakten Verfahren sind in der Lage, eine Problemstellung optimal zu lösen. Allerdings steigt mit wachsender Problemgröße der Rechenaufwand exponentiell an, sodass diese Verfahren lediglich für kleine Problemstellungen geeignet sind. Die erfolgreichsten Algorithmen, die das CVRP exakt lösen, lösen Instanzen bis zu 121 Knoten (Baldacci et al., 2010, S. 8). Das verdeutlicht wiederum, dass selbst mit heutigen Rechnerkapazitäten viele praktische Probleme noch nicht zur Optimalität in zufriedenstellender Zeit gelöst werden können. Insbesondere dann, wenn man bedenkt, dass die Tourenplanung wie beschrieben i.d.R. täglich stattfindet und somit nur eine begrenzte Zeit zur Berechnung der Lösung zur Verfügung steht.

Aufgrund dieser Anforderungen kommen beim Lösen von kombinatorischen Optimierungsproblemen oftmals heuristische Verfahren zum Einsatz. Sie versuchen nicht das Problem exakt zu lösen, sondern nur möglichst exakt, ohne eine Aussage darüber treffen zu können, ob die Problemlösung, die letztendlich ermittelt wird, wirklich die Optimallösung dargestellt. Der Vorteil heuristischer Verfahren liegt darin begründet, dass sie sehr viel weniger Rechenzeit benötigen, um zu einer guten oder sehr guten Lösung zu gelangen. Hier erkauft man sich jedoch die kürzere Ausführungsdauer durch eine gegebenenfalls schlechtere Lösungsqualität im Vergleich zur Optimallösung. Heuristische und exakte Verfahren haben gemein, dass sie problemspezifisch sind und somit nicht ohne Weiteres auf andere Problemstellungen und deren Lösung übertragen werden können. Heuristische Verfahren bestehen nach Müller-Merbach (1992, S. 290) „aus bestimmten Vorgehensregeln zur Lösungsfindung, die hinsichtlich des angestrebten Zieles und unter Berücksichtigung der Problemstruktur als sinnvoll, zweckmäßig und erfolgsversprechend erscheinen, aber nicht immer die optimale Lösung hervorbringen“. Weiterhin unterscheidet man bei

Heuristiken oftmals zwischen Eröffnungs- bzw. Konstruktions- und Verbesserungsverfahren (Domschke und Scholl, 2005, S. 80), die in Abschnitt 2.3.2 näher behandelt werden.

Metaheuristiken garantieren ebenfalls nicht, dass die Optimallösung eines kombinatorischen Optimierungsproblems gefunden wird. Sie benötigen jedoch wie Heuristiken bei großen Problemen eine sehr viel geringere Rechenzeit zum Finden einer sehr guten Lösung im Vergleich zu exakten Verfahren. Der Unterschied zu Heuristiken besteht darin, dass eine Metaheuristik nicht problemspezifisch ist, sondern sich von ihrem konzeptionellen Aufbau her auf unterschiedlichste Probleme gleichermaßen anwenden lässt.[4] Oftmals sind heuristische und/oder exakte Verfahren Bestandteil von Metaheuristiken. Dabei übernimmt die Metaheuristik die Aufgabe des Steuerns der Heuristik (Domschke und Scholl, 2005, S. 82). In Analogie dazu definieren Osman und Laporte (1996, S. 514f.) eine Metaheuristik als „an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space“. Dieses Prinzip soll bei der näheren Betrachtung verschiedener Metaheuristiken in den folgenden Abschnitten deutlich werden.

2.3.2 Heuristiken zur Lösung des CVRPs

Wie in Abschnitt 2.3.1 beschrieben, kann man bei Heuristiken im Allgemeinen zwischen Eröffnungs- bzw. Konstruktionsverfahren und Verbesserungsverfahren unterscheiden. In den folgenden Unterabschnitten wird zunächst auf die Eröffnungsverfahren eingegangen, bevor danach die Verbesserungsverfahren näher beleuchtet werden. Der Fokus liegt dabei auf Methoden für das CVRP.

2.3.2.1 Eröffnungs- und Konstruktionsverfahren

Um mit einem Lösungsverfahren für ein kombinatorisches Optimierungsproblem zu starten, ist es erforderlich, eine Ausgangslösung zu bestimmen. Dazu werden Eröffnungsverfahren verwendet.[5] Sie generieren nach definierten Regeln in möglichst kurzer Zeit eine Ausgangslösung. Je nach Verfahren, das auf die Konstruktionsphase folgt, variieren die Anforderungen an die Eröffnungsverfahren. Beispielsweise wird in der vorliegenden Arbeit von dem Eröffnungsverfahren eine gültige Ausgangslösung gefordert. Es gibt jedoch auch viele Verfahren, die keine gültige Lösung benötigen. In Abhängigkeit der Anforderungen variiert die Laufzeit des Eröffnungsverfahrens. In der Regel nimmt das Eröffnungsverfahren jedoch die geringste Zeit eines Algorithmus zur Lösung eines Optimierungsproblems in Anspruch.

Für das CVRP existieren unterschiedlichste Algorithmen, um Initiallösungen (sowohl gültige als auch ungültige) zu generieren. Eine der bekanntesten ist das Nearest-Neighbor-Verfahren. Ausgangspunkt bildet hierbei das Depot. Im nächsten Schritt wird der Kunde, der am nächsten zum Depot liegt, in die Tour aufgenommen. Danach wird der nächste Nachbar (der noch nicht bedient wird) von dem Kunden, der als letztes zur aktuellen Tour hinzugefügt wurde, in die aktuelle Tour aufgenommen, sofern die Kapazitätsrestriktion nicht verletzt ist. Sollte die Restriktion verletzt sein, wird die aktuelle Tour geschlossen und eine neue Tour geöffnet. Nun wird wieder vom Depot ausgehend der nächste Nachbar, der bisher von keiner Tour bedient wird, zur aktuell offenen Tour hinzugefügt. Das Verfahren endet,

[4] Natürlich ist es notwendig Anpassungen an einer Metaheuristik bzw. an ihren Bestandteilen vorzunehmen, damit sie auf unterschiedliche Probleme angewendet werden kann. Allerdings ändern die Anpassungen nichts an der grundsätzlichen Idee, wie die Metaheuristik im Suchverlauf vorgeht.

[5] Auch bei einem Eröffnungs- bzw. Konstruktionsverfahren handelt es sich um ein Lösungsverfahren. Allerdings folgen auf Eröffnungsverfahren zumeist weitere Lösungsverfahren, um die Ausgangslösung weiter zu optimieren.

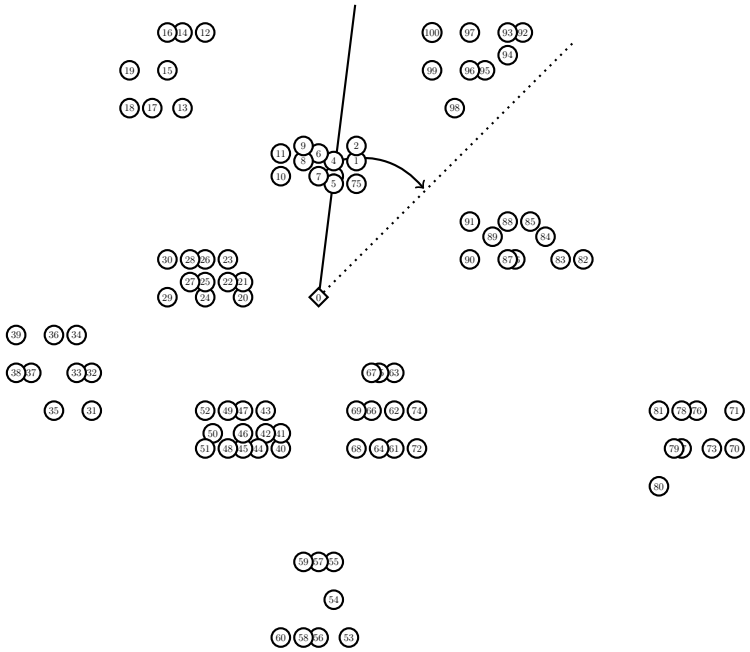


Abbildung 2.2: Sweep-Konstruktionsverfahren in Anlehnung an Dondo und Cerdá (2009, S. 516)

wenn alle Kunden bedient sind. Schon hier sind diverse Varianten denkbar. Wenn bspw. die maximale Anzahl der zur Verfügung stehenden Fahrzeuge erreicht ist, was geschieht dann mit den bis dahin nicht belieferten Kunden? Eine Möglichkeit könnte sein, sie unbedient zu lassen und darauf zu hoffen, dass sie im weiteren Ablauf des Gesamt-Algorithmus noch bedient werden. Oder unbelieferte Kunden werden ohne Berücksichtigung der Kapazitätsrestriktion in die letzte offene Tour eingefügt.

Ein weiteres bekanntes Eröffnungsverfahren ist das Sweep-Verfahren, das von Gillett und Miller (1974) stammt. Hier wird ebenfalls die Lokalität der Kunden, die auf einer Tour bedient werden, berücksichtigt. Ausgangspunkt ist eine Gerade vom Depot mit einem beliebigen Winkel (vgl. Abbildung 2.2). Diese Gerade wird um das Depot herum bewegt. Sobald die Gerade einen Kunden schneidet, wird dieser zur aktuellen Route hinzugefügt. Sollte die Kapazitätsrestriktion verletzt sein, wird die derzeitige Route geschlossen und eine neue geöffnet, in die dann der Kunde eingefügt wird. Das Verfahren endet, wenn alle Kunden bedient sind bzw. die Gerade das Depot einmal mit 360 Grad umwandert hat. Auch bei diesem Verfahren stellen sich die gleichen Fragen, die bei der Beschreibung des Nearest-Neighbour-Algorithmus diskutiert wurden. Wie oben bereits erwähnt, existieren auch hierfür verschiedenste Lösungsansätze, wie mit unbedienten Kunden umgegangen werden soll.

Ein häufig verwendeter Algorithmus zur Erstellung einer Startlösung für das CVRP stammt von Clarke und Wright (1964). Der Algorithmus basiert auf einer möglichen Kostenreduktion, die dadurch erzielt werden kann, indem zwei Routen zu einer Route zusammengelegt werden. Das Lösungsverfahren beginnt mit einer Initiallösung, in der so viele Fahrzeuge bzw. Routen enthalten sind wie Kunden. Danach werden die beiden Routen ausgewählt, bei denen die Kostenreduktion durch Zusammenlegung am höchsten ist. Angenommen, es sind die beiden Routen $(v_0, ..., v_i, v_0)$ und $(v_0, v_j, ..., v_0)$ gegeben, dann werden diese beiden zu einer Route $(v_0, ..., v_i, v_j, ...v_0)$ zusammengefügt, wenn sich eine Kostenreduktion $s_{v_i,v_j} = c_{v_i,v_0} + c_{v_0,v_j} - c_{v_i,v_j}$ ergibt, d.h. wenn s_{v_i,v_j} größer 0 ist (Laporte und Semet, 2002, S. 110). Je nach Anforderung an die zu generierende Lösung sind neben der Berechnung der Kostenreduktion

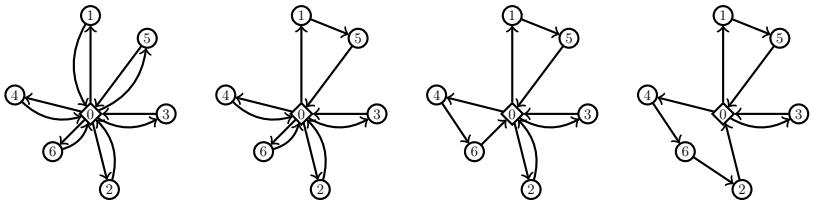


Abbildung 2.3: Clarke-and-Wright-Savings-Konstruktionsverfahren

eventuell auch Restriktionen zu berücksichtigen. In Abbildung 2.3 ist das Verfahren schematisch dargestellt. Gestartet wird mit einer Initiallösung, in der jeder Kunde genau einem Fahrzeug zugeordnet ist (siehe Abbildung 2.3(a)). Nun werden die Savings bestimmt. In Beispielfall gilt, dass $s_{1,5} > 0$, d.h. $c_{1,0} + c_{0,5} - c_{1,5} > 0$ ist. Weiterhin wird angenommen, dass keine Kapazitätsverletzung durch die Zusammenlegung der zwei Routen entsteht und somit resultiert die neue Lösung in Abbildung 2.3(b). Der Algorithmus wird so lange fortgesetzt bis entweder keine Kostenreduktion durch die Zusammenlegung von Routen mehr möglich ist oder die Kapazitätsrestriktion nicht mehr erfüllt werden kann. Eine mögliche Endlösung des Algorithmus ist in Abbildung 2.3(d) dargestellt. Zum Clarke-and-Wright-Savings-Algorithmus gibt es viele Erweiterungen, auf die an dieser Stelle nicht weiter eingegangen werden soll. Eine gute Übersicht zu den Erweiterungen sowie zu weiteren Konstruktionsalgorithmen für das CVRP findet man in dem Beitrag von Laporte und Semet (2002, S. 111-116).

2.3.2.2 Verbesserungsverfahren

Verbesserungsverfahren nutzen die Lösung, die durch ein Konstruktionsverfahren erzeugt wurde, als Ausgangsbasis für den eigenen Suchprozess. Oftmals werden Verbesserungsheuristiken im Kontext von VRPs gleichgesetzt mit Lokalsuchen. Dieser Nomenklatur wird in dieser Arbeit gefolgt. Verbesserungsverfahren haben i.d.R. gemein, dass sie sich von einer Lösung zur nächsten bewegen. Dabei versuchen sie stets, die aktuelle Lösung zu verbessern. Es handelt sich also um einen iterativen Prozess, bei dem die Startlösung s durch einen Schritt - auch Move genannt - zu einer neuen Lösung s' transformiert wird. s' dient dann als Ausgangspunkt für den nächsten Iterationsschritt, in dem versucht wird, die Lösung s' zu verbessern (Domschke und Scholl, 2005, S. 81). Verbesserungsverfahren stoppen, wenn keine Verbesserung mehr von der aktuellen Lösung erzielt werden kann, d.h., es existiert kein Move mehr, der die aktuelle Lösung in eine mit einem besseren Zielfunktionswert transformiert.

Der Move bzw. Suchoperator M, der eine Lösung s in Lösung s' transformiert, definiert gleichzeitig die Nachbarschaft $N_M(s)$ der Lösung s. $N_M(s)$ enthält alle Lösungen, die von Lösung s ausgehend, mittels Anwendung des Moves M bzw. Movetyps auf s, erreicht werden können. $N_M(s)$ enthält also „die Menge aller Nachbarlösungen“ (Domschke und Scholl, 2005, S. 81) von s. Die Art des Moves bestimmt somit direkt die Größe der Nachbarschaft. Verbesserungsverfahren besitzen die Eigenschaft, den besten Move bzw. die beste Nachbarschaftslösung zu ermitteln, die dann den Ausgangspunkt für die weitere Suche bildet. Die Anzahl der Lösungen, die bei einem Move(typ) durchsucht werden müssen, um eine Lösungstransformation letztendlich durchführen zu können, bestimmt den Rechenaufwand.

Bei Betrachtung von VRPs kann man in diesem Zusammenhang nach Laporte und Semet (2002, S. 122) zwischen Intra- und Inter-Route-Verfahren unterscheiden. Bei einem Intra-Route-Verfahren werden nur Kanten, die innerhalb einer Route liegen, entfernt und neue Kanten nur in dieser Route

hinzugefügt. Man bearbeitet im Falle des CVRPs eine Teillösung, die dem TSP entspricht. Bei einem Inter-Route-Verfahren sind mehrere Routen beteiligt, wobei sich oftmals auf zwei unterschiedliche Touren beschränkt wird. Im Folgenden finden sich detaillierte Beschreibungen einzelner Moves (sowohl Intra- als Inter-Route-Moves), die in dieser Arbeit Anwendung finden. Weitere Informationen zu Verbesserungsverfahren finden sich bspw. in dem Beitrag von Laporte und Semet (2002).

Wie bereits beschrieben, wird ein Operator bzw. Move jeweils auf eine bestehende Lösung angewendet und transformiert diese zu einer neuen Lösung. Eine mögliche Vorgehensweise besteht darin, dass man den ersten Knoten in der ersten Route betrachtet und bewertet anhand dessen alle möglichen Move-Kombinationen, die mit dem gewählten Knoten und der bestehenden Lösung möglich sind. Eine andere Möglichkeit ist, dass man sog. erzeugende Kanten (auch generierende Kanten oder Generatorkanten genannt) betrachtet, aus denen sich ein Move ergibt (Toth und Vigo, 2003, S. 339). Somit ist durch die Erzeugerkante der zu bewertende Move determiniert. Mit einem solchen Vorgehen ist es einfach möglich, lange Kanten, die vermutlich nicht in sehr guten Lösungen eines CVRPs enthalten sind, zu Beginn bereits zu entfernen und somit weniger Operatoren bzw. Erzeugerkanten pro Durchlauf bewerten zu müssen. Auf Letzterem aufbauend werden im Rahmen dieser Arbeit sechs Operatoren, die im Folgenden vorstellt werden, implementiert.

Relocate-Operator

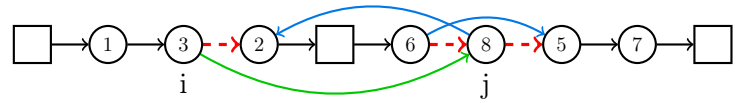
Die Grundgedanke des Relocate-Operators besteht darin, einen Knoten aus einer Route an eine beliebige Stelle in einer anderen Tour zu setzen (Bräysy und Gendreau, 2005, S. 111). Grundsätzlich handelt es sich bei diesem Move also um einen Inter-Route-Move. Allerdings ist es genauso möglich, die Position eines Knoten nur innerhalb einer Route zu verändern. Da die Lösung eines CVRP in dieser Arbeit als eine Gesamttour[6] gespeichert ist (vgl. Kapitel 3.1.1), werden beide Möglichkeiten berücksichtigt, d.h. ein Relocate-Move kann in diesem Beitrag sowohl innerhalb als auch zwischen zwei Routen ausgeführt werden.

In Abbildung 2.4 ist ein Relocate-Move zwischen zwei verschiedenen Routen dargestellt. Ausgehend von einer Kantenliste werden die Indices auf der aktuellen Gesamttour der zwei Knoten, die in der neuen Lösung miteinander verbunden werden sollen, bestimmt. Im Beispielfall sollen die Knoten $v_i = 3$ und $v_j = 8$ verbunden werden. Die erzeugende Kante des Moves (in den folgenden Abbildungen grün dargestellt) ist demnach die Kante mit den Endpunkten 3 und 8 (ihre Indices werden im Folgenden als i bzw. j bezeichnet). Daraus resultiert, dass Knoten 8 hinter Knoten 3 eingefügt werden muss, da die Generatorkante Bestandteil der neuen Lösung sein muss. Somit ergibt sich, dass die Kanten $[v_i, v_{i+1}]$, $[v_{j-1}, v_j]$ und $[v_j, v_{j+1}]$ aus der aktuellen Lösung gelöscht werden und gleichzeitig die Kanten $[v_i, v_j]$, $[v_j, v_{i+1}]$ und $[v_{j-1}, v_{j+1}]$ Bestandteil der neuen Lösung sind.

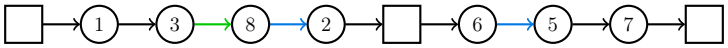
Damit lässt sich die Veränderung der Distanz von der ursprünglichen Lösung zur neuen Lösung in $O(1)$ berechnen, wozu folgende Formel zum Einsatz kommt:

$$c_\Delta = -\overbrace{(c_{v_i,v_{i+1}} + c_{v_{j-1},v_j} + c_{v_j,v_{j+1}})}^{=c_a} + \underbrace{(c_{v_i,v_j} + c_{v_j,v_{i+1}} + c_{v_{j-1},v_{j+1}})}_{=c_n} \tag{2.10}$$

[6] Im Englischen wird diese Speicherweise als „giant tour“ bezeichnet, weshalb im Folgenden auch die deutsche Übersetzung „große Tour“ verwendet wird.



(a) Vorher



(b) Nachher

Abbildung 2.4: Relocate-Operator - Inter-Route

Die i-te bzw. j-te Position lässt sich mittels eines einfachen Lookups bewerkstelligen, da ein Array vorgehalten wird, in dem vermerkt ist, an welcher Position sich in der aktuellen Lösung ein Knoten befindet. Damit lassen sich direkt die anderen Knoten, die an dem Move beteiligt sind, über die Indices ermitteln. c_a enthält die Kosten der Kanten, die aus der aktuellen Lösung entfernt werden und c_n die Kosten der Kanten, die Bestandteil der neuen Lösung sind. Die Differenz dieser beiden ergibt c_Δ, das die Veränderung der Gesamtdistanz bei Anwendung des Moves angibt. Wenn c_Δ kleiner 0 ist, ist die neue Lösung besser als die aktuelle, wenn c_Δ größer 0 ist, verschlechtert sich die Lösungsgüte. Sollte c_Δ 0 ergeben, verändert sich die Gesamtdistanz nicht; allerdings kann sich die Anordnung der Knoten, also die Lösung, durch Ausführung des Moves trotzdem ändern. Gerade bei sehr strukturierten Benchmarkinstanzen dürfte dieser Fall häufiger eintreten. Da im Falle des Relocate-Operators drei Kanten entfernt bzw. hinzugefügt werden, handelt es sich bei dem Operator um einen Spezialfall des 3-Opt (Reimann et al., 2003, S. 305).

Neben der Berechnung der Distanzveränderung ist ein weiterer wichtiger Bestandteil die Bestimmung der Kapazitätsveränderung der beteiligten Routen. Dazu wird folgende Formel angewendet:

$$C^n_{r_{v_i}} = C^a_{r_{v_i}} + d_{v_j} \tag{2.11}$$

$$C^n_{r_{v_j}} = C^a_{r_{v_j}} - d_{v_j} \tag{2.12}$$

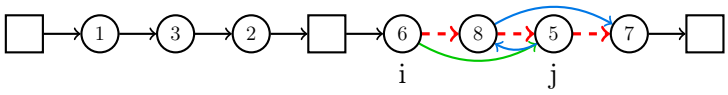
m entspricht der Anzahl der Fahrzeuge bzw. Routen. Die Kapazität jeder einzelnen Route C_r mit $r \in \{1, ..., m\}$ wird in der vorliegenden Implementierung in einem Array gespeichert. r_{v_i} gibt dabei die Route bzw. deren ID, auf der sich ein Knoten befindet, an. d_{v_i} mit $v_i \in \{1, ..., n\}$ liefert die nachgefragte Menge von einem Knoten. Beim Relocate-Operator ergibt sich somit die Veränderung der Kapazität dadurch, dass der Route von Knoten v_i die Nachfrage von Knoten v_j hinzugefügt wird und im Umkehrschluss der Route von Knoten v_j die Nachfrage von Knoten v_j abgezogen wird. Alle verwendeten Daten werden im Arbeitsspeicher vorgehalten und es ist nicht nötig, die Berechnungen bspw. zur Bestimmung der Kapazität einer Route komplett neu durchzuführen.

Da im Rahmen dieser Arbeit nur gültige Moves akzeptiert werden bzw. nur mit gültigen Lösungen gearbeitet wird, ist lediglich die Veränderung der Kapazität der Route von Knoten v_i zu bestimmen.[7] Es gilt, dass die aktuelle Lösung immer gültig ist. Wenn man also aus einer Route einen Knoten entfernt, so ist diese im Falle des CVRPs immer noch gültig, da die benötigte Kapazität weiterhin

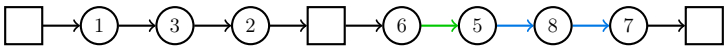
[7] Hierbei wird sich nur auf die Bewertung des Moves bezogen. Wenn der Operator schließlich auf der aktuellen Lösung ausgeführt wird, ist auch die Kapazitätsveränderung von der Route von Knoten v_j zu bestimmen.

kleiner ist als die maximal zur Verfügung stehende Kapazität C. Daher muss nur die Kapazität der Route von Knoten v_i mittels Gleichung 2.11 überprüft werden. Sollte die neue Kapazität die maximal verfügbare Kapazität pro Fahrzeug übersteigen, wird die betrachtete Generatorkante für die aktuelle Ausgangslösung verworfen. Die Zeitaufwand zur Bestimmung der neuen Kapazität beträgt $O(1)$.

Abbildung 2.5 zeigt den Relocate-Move innerhalb einer Route (Intra-Route). Die Distanzveränderung ergibt sich in Analogie zum Inter-Route-Fall, allerdings kann nun auf die Berechnung der Kapazitätsveränderung verzichtet werden. Es wird der betrachteten Route weder ein Knoten hinzugefügt noch entfernt, sondern es werden nur die Positionen der Knoten innerhalb der Route getauscht und es findet keine Veränderung der Kapazität statt.



(a) Vorher

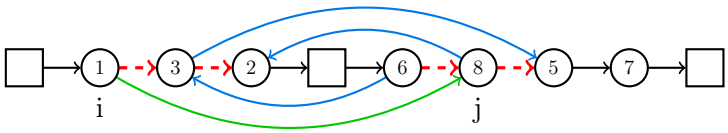


(b) Nachher

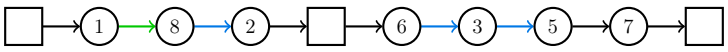
Abbildung 2.5: Relocate-Operator - Intra-Route

Swap-Operator

Der Swap-Operator - auch Exchange-Operator genannt (Bräysy und Gendreau, 2005, S. 111) - arbeitet in Analogie Relocate-Operator. Es gilt hierbei, dass zwei Knoten bzw. Kunden aus einer großen Tour ihre Position tauschen. In Abbildung 2.6 ist der Move dargestellt, wenn die Positionen zwischen zwei verschiedenen Routen getauscht werden.



(a) Vorher



(b) Nachher

Abbildung 2.6: Swap-Operator - Inter-Route

Es wird ersichtlich, dass man die Tauschoperation theoretisch durch zwei Relocate-Moves erreichen kann. Allerdings ergibt sich in diesem Falle die Problematik der Kapazitätsrestriktion. Wenn man annimmt, dass die liefernden Fahrzeuge nahe der maximal erlaubten Kapazität ausgelastet sind, so können sie keinen einzelnen Kunden mehr aufnehmen, ohne die Restriktion zu verletzen. Da im Rahmen dieser Arbeit nur gültige Lösungen akzeptiert werden, ist es mit zwei sequentiell ausgeführten Relocate-Moves nicht zwangsläufig möglich, jeden Zustand, den auch ein Swap-Move erzeugen kann, zu

erreichen. Bei dem Swap-Move reduziert sich bei jeder der beteiligten Routen die Kapazitätsauslastung um den entfernten Knoten und schafft eventuell wieder Platz für einen weiteren Kunden. Aufgrund dessen wird im Rahmen der Arbeit sowohl ein Relocate- als auch ein Swap-Move verwendet. Falls von einem Algorithmus temporär ungültige Lösungen erlaubt werden, muss zunächst diskutiert werden, ob ein Relocate- und Swap-Operator implementiert werden sollen oder ob es ausreicht, nur einen Relocate-Operator zu implementieren.

Wie in Abbildung 2.6 erkennbar, baut sich der Swap-Move in diesem Beitrag ebenfalls mit einer erzeugenden Kante von Knoten 1 zu Knoten 8 auf. Es wird also Knoten $v_j = 8$ in Route r_{v_i} eingefügt und Knoten $v_{i+1} = 3$ in Route r_{v_j}. Daraus ergibt sich folgende Distanzveränderung:

$$\begin{aligned} c_\Delta &= -\overbrace{(c_{v_i,v_{i+1}} + c_{v_{i+1},v_{i+2}} + c_{v_{j-1},v_j} + c_{v_j,v_{j+1}})}^{=c_a} \\ &\quad + \underbrace{(c_{v_i,v_j} + c_{v_j,v_{i+2}} + c_{v_{j-1},v_{i+1}} + c_{v_{i+1},v_{j+1}})}_{=c_n} \end{aligned} \tag{2.13}$$

Der erste Teil der Formel c_a ermittelt die Kosten der Kanten, die aus der aktuellen Lösung entfernt werden sollen, und c_n bestimmt die Kosten der Kanten, die Teil der neuen Lösung sind. Auch hier gilt wieder, dass ein c_Δ kleiner 0 eine Kostenreduktion der neuen Lösung im Vergleich zur ursprünglichen angibt, ein c_Δ größer 0 eine Verschlechterung darstellt und c_Δ gleich 0 keine Kostenveränderung mit sich bringt. Da bei der Operation vier Kanten beteiligt sind, lässt sich der Swap-Move dem 4-Opt zuordnen (Reimann et al., 2003, S. 305).

Die Kapazitätsbestimmung erfolgt ebenfalls in Anlehnung an den Relocate-Operator, jedoch mit den bereits weiter oben beschriebenen Erweiterungen, dass eine Verringerung und Erhöhung der Kapazität in beiden beteiligten Routen stattfindet. Die Kapazität der Routen berechnet sich folgendermaßen:

$$C^n_{r_{v_i}} = C^a_{r_{v_i}} + d_{v_j} - d_{v_{i+1}} \tag{2.14}$$

$$C^n_{r_{v_j}} = C^a_{r_{v_j}} - d_{v_j} + d_{v_{i+1}} \tag{2.15}$$

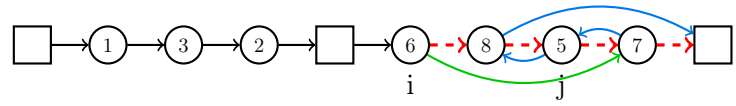
Die ursprüngliche Kapazität der Route von Knoten v_i reduziert sich um die Nachfrage von Knoten v_{i+1} und erhöht sich um die Nachfrage von Knoten v_j. In Analogie dazu ergibt sich die Kapazität $C^n_{r_{v_j}}$. Der Zeitaufwand zur Bestimmung der neuen Kapazitäten und somit auch zur Bestimmung, ob die Restriktion verletzt ist, beträgt $O(1)$.

Bei einem Intra-Route-Move (siehe Abbildung 2.7) ergeben sich die gleichen Schlussfolgerungen, die im Rahmen des Relocate-Moves gemacht werden, d.h., eine Überprüfung der Kapazitäten ist nicht nötig, da keine Knoten in die Route hinzugefügt werden.

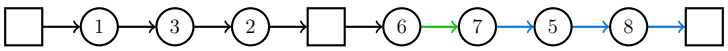
Or-Opt-Operator

Der Or-Opt-Operator wird von Or (1976) vorgeschlagen. Er geht wieder einen Schritt im Vergleich zum Swap-Move zurück, hin zur Klasse des 3-Opt (Reimann et al., 2003, S. 305). Bei Anwendung des Operators werden drei Kanten aus der Ausgangslösung entfernt und drei neue Kanten hinzugefügt. Die Idee des Or-Opt-Operators ist, dass eine Sequenz von Kunden in einer Route ausgeschnitten und an anderer Stelle wieder eingefügt wird. Die Sequenz von Kunden, die entfernt und wieder eingefügt wird, kann einen bis beliebig viele Kunden enthalten[8], wobei hier nur der Fall betrachtet wird, in dem sich

[8] Die maximale Anzahl von Kunden der Sequenz kann natürlich nicht die Anzahl der Kunden, die überhaupt beliefert werden müssen, übersteigen.



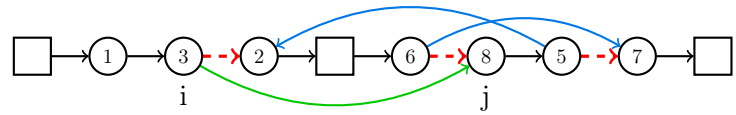
(a) Vorher



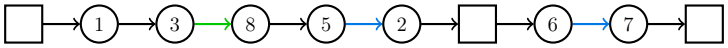
(b) Nachher

Abbildung 2.7: Swap-Operator - Intra-Route

die Kundensequenz vollständig auf einer Route befindet[9].



(a) Vorher



(b) Nachher

Abbildung 2.8: Or-Operator - Inter-Route

Abbildung 2.8 zeigt, wie eine Sequenz von zwei Kunden an eine andere Position verschoben wird. Auch in diesem Zusammenhang kann wieder zwischen Inter- und Intra-Route unterschieden werden. Die Distanzveränderung für den in der Abbildung dargestellten Fall ergibt sich mittels folgender Formel:

$$\begin{aligned} c_\Delta &= -(c_{v_i,v_{i+1}} + c_{v_{j-1},v_j} + c_{v_{j+1},v_{j+2}}) \\ &\quad +(c_{v_i,v_j} + c_{v_{j+1},v_{i+1}} + c_{v_{j-1},v_{j+2}}) \end{aligned} \tag{2.16}$$

Nimmt man nun an, dass nicht nur zwei Kunden, also v_j und der erste Kunde hinter Knoten v_j verschoben werden, sondern l Knoten hinter Knoten v_j, so resultiert daraus die allgemeingültige Formel 2.17. Setzt man $l = 1$ so ergibt sich der in Abbildung 2.8 dargestellte Spezialfall, dass zwei Knoten verschoben werden.

$$\begin{aligned} c_\Delta &= -\overbrace{(c_{v_i,v_{i+1}} + c_{v_{j-1},v_j} + c_{v_{j+l},v_{j+l+1}})}^{=c_a} \\ &\quad +\underbrace{(c_{v_i,v_j} + c_{v_{j+l},v_{i+1}} + c_{v_{j-1},v_{j+l+1}})}_{=c_n} \end{aligned} \tag{2.17}$$

Wie schon bei den zuvor vorgestellten Operatoren bildet wieder eine erzeugende Kante den Ursprung für den vollständigen Move, wenn zuvor l festgelegt wurde. c_Δ ergibt sich in Analogie aus der Differenz von c_a und c_n. Auch hier gilt für c_Δ größer 0, dass eine Kostensteigerung, für c_Δ gleich, dass keine

[9] Somit ergibt sich eine noch stärkere Einschränkung für die Anzahl der Kunden, die verschoben werden. Die Anzahl entspricht maximal der Anzahl der Knoten, die in der entsprechenden Route, aus der die Kundensequenz entfernt werden soll, enthalten sind.

Kostenveränderung und für c_Δ kleiner 0, dass eine Kostensenkung im Vergleich zur Ursprungslösung vorliegt.

Die Kapazitätsveränderung ergibt sich folgendermaßen:

$$C^n_{r_{v_i}} = C^a_{r_{v_i}} + \sum_{m=j}^{j+l} d_{v_m} \tag{2.18}$$

$$C^n_{r_{v_j}} = C^a_{r_{v_j}} - \sum_{m=j}^{j+l} d_{v_m} \tag{2.19}$$

Da nur Knoten in Route r_{v_i} eingefügt werden, ist es zur Bestimmung, ob die Kapazitätsrestriktion verletzt wird, nur nötig, Formel 2.18 anzuwenden. Für den in Abbildung 2.8 dargestellten Fall, dass $l = 1$, ergibt sich folgende spezifische Formel zur Überprüfung der Kapazitätsverletzung:

$$C^n_{r_{v_i}} = C^a_{r_{v_i}} + d_{v_j} + d_{v_{j+1}} \tag{2.20}$$

Damit lässt sich die Kapazitätsbeschränkung für den Spezialfall in $O(1)$ und für den allgemeinen Fall in $O(l)$ überprüfen.

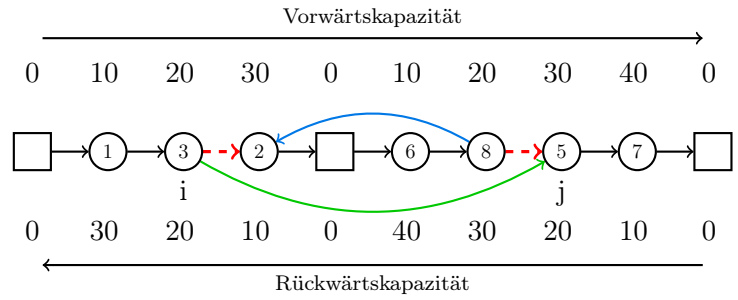
Die vorliegende Arbeit beschränkt sich aus praktischen Überlegungen heraus lediglich auf den statischen Fall $l = 1$. Hier kommt ein ähnliches Argument zum Tragen, das schon im Rahmen der Beschreibung des Swap-Operators erwähnt wird. Je mehr Knoten von einer Route in eine andere verschoben werden, ohne dass in der Route, in der die Knotensequenz eingefügt wird, auch Knoten entfernt werden, umso häufiger wird man an die Kapazitätsgrenzen stoßen, wenn l größer gewählt ist und das Fahrzeug stark ausgelastet ist. Es wird dann zunehmend schwieriger, gültige Or-Opt-Moves zu finden, wenn viele Kunden verschoben werden sollen. Aufgrund dieser Überlegungen beschränkt sich die Implementierung dieser Arbeit auf eine maximale Sequenz von zwei Kunden, die verschoben werden.

2-Opt*-Operator

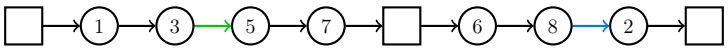
Der 2-Opt*-Operator wird von Potvin und Rousseau (1995) eingeführt. Im Gegensatz zu den bisher vorgestellten Operatoren wird der 2-Opt*-Move nur zwischen zwei Routen angewendet, d.h., der Operator ist ein reiner Inter-Route-Move. Das resultiert aus der grundlegenden Idee des 2-Opt*, dass zwei Routenenden miteinander verbunden und dabei die ursprüngliche Richtung der Route bzw. die Reihenfolge der Knoten beibehalten werden soll.[10] Dazu wird in den beiden beteiligten Routen jeweils eine Kante entfernt und der neuen Lösung werden insgesamt zwei neue Kanten hinzugefügt. Der 2-Opt*-Move ist also in naher Verwandtschaft zu dem 2-Opt-Move, der ab Seite 19 vorgestellt wird.

In Abbildung 2.9 ist ein möglicher 2-Opt*-Move dargestellt. Man kann erkennen, dass wieder eine erzeugende Kante mit den Endpunkten 3 und 5 den Ausgangspunkt für den Operator bildet. Das Einfügen der erzeugenden Kante fordert, dass die Kanten zwischen den Knoten 3 und 2 sowie zwischen 8 und 5 entfernt werden müssen und die Kante zwischen Knoten 8 und 2 neben der Erzeugerkante in die neue Lösung eingefügt wird.

[10] Für das CVRP ist die Richtungserhaltung der Kanten, die nicht gelöscht werden, weniger von Bedeutung. Der Operator spielt seine Stärke insbesondere bei Problemklassen aus, bei denen es nicht unerheblich ist - z.B. aufgrund von Restriktionen - ob zunächst bspw. Kunde A und dann Kunde B oder umgekehrt bedient wird. Ein prominentes Beispiel dafür ist das VRP with Time Windows (VRPTW), bei dem durch die Zeitfensterrestriktion die Bedienung von A und dann B gültig sein kann und die Umkehrung von B vor A ungültig, weil dadurch das Zeitfenster von A verletzt wird.



(a) Vorher



(b) Nachher

Abbildung 2.9: 2-Opt*-Operator - Inter-Route

Die Kostenreduzierung resultiert aus:

$$c_\Delta = -\overbrace{(c_{v_i,v_{i+1}} + c_{v_{j-1},v_j})}^{=c_a} + \underbrace{(c_{v_i,v_j} + c_{v_{j-1},v_{i+1}})}_{=c_n} \tag{2.21}$$

Es gelten die gleichen Aussagen für die Ausprägungen von c_Δ wie zuvor.

Anders verhält es sich hingegen bei Berechnung der Kapazität der beteiligten Routen. Es ist nun nicht mehr wie bei den bisher vorgestellten Operatoren möglich, die Kapazität lediglich durch Subtraktion und Addition von nachgefragten Mengen zu bestimmen, sondern es muss auf weitere Datenstrukturen zurückgegriffen werden, um eine effiziente Kapazitätsberechnung zu realisieren.[11] Dazu kommen die beiden Datenstrukturen Vorwärts- und Rückwärtskapazität zum Einsatz, wie sie in Abbildung 2.9(a) dargestellt sind. Es wird für jeden Knoten v_i der aktuellen Lösung ein Wert für die Vorwärtskapazität $\overrightarrow{d}_{v_i}$ und die Rückwärtskapazität $\overleftarrow{d}_{v_i}$ berechnet. Die Vorwärtskapazität eines Knoten ergibt sich aus der Summe der nachgefragten Menge seiner Vorgängerknoten in seiner Route und seiner eigenen nachgefragten Kapazität, wohingegen die Rückwärtskapazität die umgekehrte Richtung betrachtet. Letztere ergibt sich aus der Summe der nachgefragten Menge der nachgelagerten Knoten des zu betrachtenden Knotens in seiner Route zuzüglich seiner eigenen nachgefragten Menge. Um das Vorgehen an einem Beispiel zu verdeutlichen, soll angenommen werden, dass die Knoten, die in Abbildung 2.9(a) dargestellt sind, alle eine Nachfrage von zehn Einheiten besitzen. Um nun die Vorwärtskapazität $\overrightarrow{d}_{v_i}$ eines jeden Knoten zu bestimmen, beginnt man mit dem ersten Knoten auf der linken Seite und durchläuft die Gesamttour. Knoten 1 hat keinen Vorgänger, wodurch sich für ihn $\overrightarrow{d}_1 = 0 + 10 = 10$, d.h. nur seine eigene Nachfragemenge, ergibt. Knoten 3 hat als Vorgänger Knoten 1 und somit resultiert $\overrightarrow{d}_3 = 10 + 10 = 20$. Analog dazu ergibt sich für Knoten 2 $\overrightarrow{d}_2 = 20 + 10 = 30$. Sobald in der großen Tour das Depot erreicht ist, wird die Vorwärtskapazität zurückgesetzt und die nächste Route beginnt wieder bei 0 (siehe dazu Knoten 6, 8, 5 und 7). Die Rückwärtskapazität ergibt sich in Analogie dazu.

[11] Möglich ist die Berechnung der Kapazitäten durch Additionen und Subtraktionen nach wie vor. Gemeint ist in diesem Zusammenhang jedoch die effiziente Berechnung der Kapazität. Da sich zu Beginn nicht vorhersagen lässt, wie viele Knoten jeweils beteiligt sind, müsste man alle Knoten der Routen betrachten und ihre nachgefragte Menge aufaddieren, was zu einem schlechten Laufzeitverhalten führt.

An Knoten 8 aus der Abbildung soll beispielhaft die Berechnung der Rückwärtskapazität dargestellt werden. Die Summe der nachgefragten Menge der Knoten, die auf Knoten 8 in der Route folgen, ist 20 (beide nachgelagerten Knoten fragen jeweils eine Menge von 10 Einheiten nach) und seine eigene Nachfrage beträgt 10 Einheiten, womit sich $\overleftarrow{d}_8 = 20 + 10 = 30$ ergibt.

Mit Hilfe dieser beiden Variablen lässt sich die neue Kapazität der beteiligten Routen mit einem Zeitaufwand von $O(1)$ wie folgt berechnen:

$$C^n_{r_{v_i}} = \overrightarrow{d}_{v_i} + \overleftarrow{d}_{v_j} \tag{2.22}$$

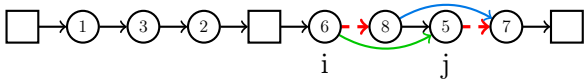
$$C^n_{r_{v_{j-1}}} = \overrightarrow{d}_{v_{j-1}} + \overleftarrow{d}_{v_{i+1}} \tag{2.23}$$

Im Falle von $C^n_{r_{v_i}}$ bedeutet das, dass man die Kapazität, die bis zum Knoten v_i (Vorwärtskapazität) anfällt, addiert mit der Kapazität, die bis zum Knoten v_j vom Ende seiner Route her betrachtet nachgefragt wird (Rückwärtskapazität). Analog dazu berechnet sich $C^n_{r_{v_{j-1}}}$ indem man die Vorwärtskapazität von Knoten v_{j-1} mit der Rückwärtskapazität von Knoten v_{i+1} addiert.

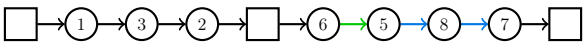
2-Opt-Operator

Im Gegensatz zu dem reinen Inter-Route-Charakter des 2-Opt*-Operators kann der 2-Opt-Operator auch innerhalb einer Tour angewendet werden. Ihm liegt die gleiche Idee, wie die des 2-Opt*-Moves zu Grunde mit dem Unterschied, dass die Richtungserhaltung der Knoten nicht gegeben ist. Das heißt, ebenso wie im Falle des 2-Opt*-Operators werden zwei Kanten aus der aktuellen Lösung entfernt und zwei andere eingefügt, woraus schließlich die neue Lösung resultiert.

Abbildung 2.10 stellt den 2-Opt-Operator innerhalb einer Route dar. Man kann erkennen, dass bei diesem Move wieder eine erzeugende Kante zwischen v_i und v_j den Ursprung bildet. Weiterhin sieht man, dass die Reihenfolge der Knoten v_{i+1} bis v_j umgekehrt wird, was jedoch im Falle des CVRP keine Auswirkung hat.[12]



(a) vorher



(b) nachher

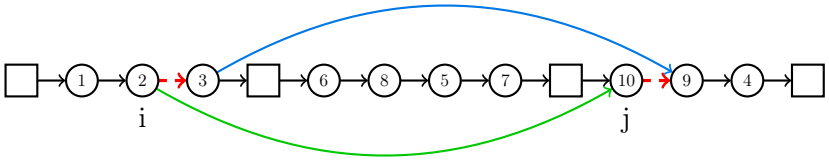
Abbildung 2.10: 2-Opt-Operator - Intra-Route

In Abbildung 2.11 ist der 2-Opt-Operator dargestellt, der mehrere Routen umfasst. Das Vorgehen ist dabei in Analogie zu dem Intra-Route-Verfahren, d.h., es werden z.T. ganze Routen innerhalb der

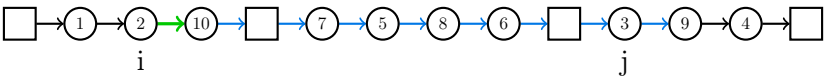
[12] Strenggenommen führt die Richtungsumkehrung dazu, dass sehr viel mehr Kanten entfernt und wieder eingefügt werden. So werden in dem Beispiel in Abbildung 2.10 die Kanten zwischen Knoten 8 und Knoten 5 ersetzt durch die Kante von Knoten 5 nach 8. Diese Umkehrung hat jedoch, wenn man lediglich die Distanz betrachtet, keine Auswirkung auf symmetrische Probleme, d.h., die Distanz von 8 nach 5 entspricht der Distanz von 5 zu 8. Die Richtungsumkehrung macht sich erst bei Betrachtung asymmetrischer VRPs oder weiterer Restriktionen (z.B. Zeitfenster) bemerkbar.

bestehenden Lösung umgekehrt.[13] Da in dieser Arbeit jedoch das symmetrische CVRP betrachtet wird, hat die Umkehrung keine Auswirkung auf die Distanz. Die Distanzveränderung der Moves ergibt sich demnach aus folgender Formel:

$$c_\Delta = -\overbrace{(c_{v_i,v_{i+1}} + c_{v_j,v_{j+1}})}^{=c_a} + \underbrace{(c_{v_i,v_j} + c_{v_{i+1},v_{j+1}})}_{=c_n} \tag{2.24}$$



(a) Vorher



(b) Nachher

Abbildung 2.11: 2-Opt-Operator - Inter-Route

Der Unterschied zwischen den Inter- und Intra-Route-2-Opt-Moves ergibt sich demnach lediglich in der Berechnung der Kapazitätsveränderung, d.h., im Falle eines Intra-Route-Moves ist keine Neuberechnung der Kapazitäten vonnöten, wohingegen sie sich im Falle eines Inter-Route-Moves verändern können. So errechnet sich die Auslastung von Route r_{v_i} mit Hilfe von Formel 2.25 und die von Route r_{v_j} mittels Gleichung 2.26. Hier wird in Analogie zum 2-Opt*-Operator wieder auf die Vorwärts- und Rückwärtskapazitäten zurückgegriffen, wodurch die Veränderungen mit einem Rechenaufwand von $O(1)$ bestimmt werden können.

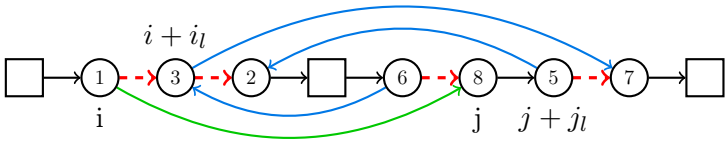
$$C^n_{r_{v_i}} = \overrightarrow{d}_{v_i} + \overrightarrow{d}_{v_j} \tag{2.25}$$

$$C^n_{r_{v_{j-1}}} = \overleftarrow{d}_{v_{i+1}} + \overleftarrow{d}_{v_{j+1}} \tag{2.26}$$

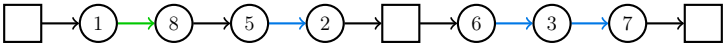
Cross-Exchange-Operator

Im Gegensatz zu den beiden zuletzt vorgestellten Operatoren, bei denen zwei Routenenden zusammengesetzt werden, wird beim Cross-Exchange-Operator eine Menge an Kunden zwischen zwei Routen ausgetauscht. Die Idee zu dem Operator stammt von Taillard et al. (1997), wobei die Autoren den Move im Kontext eines Tourenplanungsproblems mit Zeitfenstern vorschlagen. Konkret werden zwei Routen selektiert, die an dem Move beteiligt sind, danach wird bestimmt, wie viele zusammenhängende Kunden der einen Route mit wie vielen zusammenhängenden Kunden der anderen Route substituiert werden. Die beiden Kundensequenzen werden dann ausgetauscht. Auch hier liegt wie schon im Falle des Swap-Operators ein Spezialfall des 4-Opt vor, da vier Kanten aus der Lösung entfernt und vier neue

[13] Die Umkehrung einer unbeteiligten Route findet nur dann statt, wenn sie sich in der großen Tour zwischen den beiden Routen befindet, die direkt an dem Move beteiligt sind.

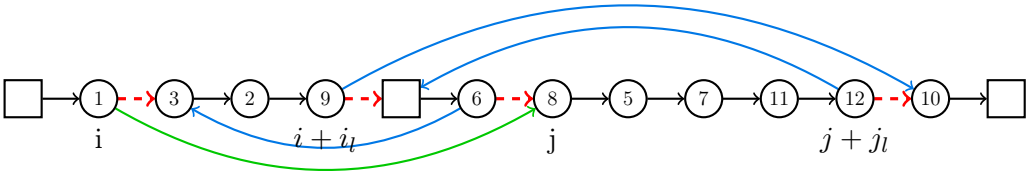


(a) Vorher

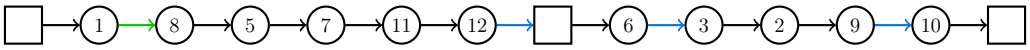


(b) Nachher

Abbildung 2.12: 1-2-Cross-Exchange-Operator - Inter-Route



(a) Vorher



(b) Nachher

Abbildung 2.13: 3-5-Cross-Exchange-Operator - Inter-Route

hinzugefügt werden. Die vorliegende Implementierung umfasst nur den Fall, dass Kundensequenzen zwischen zwei Routen getauscht werden, d.h., es handelt sich um einen Inter-Route-Move. Es ist genauso denkbar, dass der Operator für Intra-Route-Moves angepasst wird.

In Abbildung 2.12 und 2.13 ist der Sachverhalt beispielhaft für einen 1-2-Cross-Exchange und einen 3-5-Cross-Exchange dargestellt. 1-2 bedeutet hierbei, dass aus einer Route eine Kundensequenz der Länge eins mit einer Kundensequenz der Länge zwei aus einer anderen Route getauscht wird. Allgemein bedeutet in dieser Arbeit ein $x - y-$Cross-Exchange, dass x zusammenhängende Kunden einer Route mit y zusammenhängenden Kunden einer anderen Route ausgetauscht werden. Im Falle des 1-2-Cross-Exchange werden die Kundensequenzen $8 \rightarrow 5$ und 3 ausgetauscht, wohingegen im Falle des 3-5-Cross-Exchange die Kundensequenzen $8 \rightarrow 5 \rightarrow 7 \rightarrow 11 \rightarrow 12$ und $3 \rightarrow 2 \rightarrow 9$ ausgetauscht werden. Die Sequenzen werden an der jeweiligen Stelle der anderen Sequenz eingefügt. Wie man in den Abbildungen erkennen kann, ist der Move richtungserhaltend. Die Veränderung der Distanz bzw. der Kosten bei Anwendung des Moves ergibt sich in Analogie zum Swap-Operator:

$$c_\Delta = -\overbrace{(c_{v_i,v_{i+1}} + c_{v_{i+i_l},v_{i+i_l+1}} + c_{v_{j-1},v_j} + cv_{j+j_l}, v_{j+j_l+1})}^{c_a} + \underbrace{(c_{v_i,v_j} + c_{v_{j+j_l},v_{i+i_l+1}} + c_{v_{j-1},v_{i+1}} + c_{v_{i+i_l},v_{j+j_l+1}})}_{c_n} \tag{2.27}$$

Dabei gibt i_l die Länge der Sequenz, die aus der Route von Knoten v_i entfernt werden soll, und $j_l + 1$ die Länge der Sequenz, die aus der Route von Knoten v_j entnommen werden soll, an.

Schließlich gilt es noch, die neuen Kapazitäten der so resultierenden Routen zu bestimmen. Dazu kommen folgende Formeln zum Einsatz:

$$C^n_{r_{v_i}} = \overrightarrow{d}_{v_i} + \overleftarrow{d}_{v_{i+i_l+1}} + \sum_{m=j}^{j+j_l} d_{v_m} \tag{2.28}$$

$$C^n_{r_{v_{j-1}}} = \overrightarrow{d}_{v_{j-1}} + \overleftarrow{d}_{v_{j+j_l+1}} + \sum_{m=i+1}^{i+i_l} d_{v_m} \tag{2.29}$$

Die Kapazität von Route r_{v_i} ergibt sich aus der Vorwärtskapazität $\overrightarrow{d}_{v_i}$ zuzüglich der Rückwärtskapazität $\overleftarrow{d}_{v_{i+i_l+1}}$ und der Summe $\sum_{m=j}^{j+j_l} d_{v_m}$ der Nachfrage, die sich aus der Knotensequenz, die neu in die Route eingefügt wird, ergibt. Das heißt, in Route r_{v_i} werden die Knoten vor v_i unberührt belassen und übernommen, weshalb hier keine Kapazitätsänderung vorliegt und die Auslastung bis zu diesem Knoten mittels der Vorwärtskapazität bestimmt werden kann. Gleiches gilt für die Knoten ab v_{i+i_l+1}, die ebenfalls ohne Änderung übernommen werden und somit mittels der Rückwärtskapazität ihre benötigte Nachfrage ermittelt werden kann. Schließlich muss noch für die einzufügende Knotensequenz die Nachfragemenge berechnet werden, wodurch sich ein Rechenaufwand zur Bestimmung der Kapazität von Route r_{v_i} als $O(j_l)$ ergibt. Analog dazu beträgt der Rechenaufwand bei Betrachtung von Route $r_{v_{j-1}}$ $O(i_l)$.

Der hier vorgestellte Cross-Exchange deckt somit einige Fälle der Operatoren, die schon auf andere Weise implementiert wurden, ab. So entspricht ein 0-1-Cross-Exchange dem Relocate-Operator, ein 1-1-Cross-Exchange dem Swap-Operator und ein 0-2-Cross-Exchange dem Or-Opt-Operator.[14]

2.3.3 Metaheuristiken

Ein großer Nachteil der Verbesserungsheuristiken ist die Tatsache, dass sie sich sehr schnell auf ein lokales Extremum hinzubewegen. Es hängt dabei sehr stark von der Initiallösung ab, in welchem lokalen Optimum die Verbesserungsheuristik endet. In Abbildung 2.14 ist das lokale Optimum bzw. der Lösungsraum schematisch dargestellt. Das gefundene lokale Optimum (■) befindet sich noch weit vom globalen Optimum (•) entfernt. Um diesen Nachteil bzw. um das „Steckenbleiben“ im lokalen Extremum abzuschwächen, werden Metaheuristiken eingesetzt. Wie bereits geschrieben, steuern sie Heuristiken bzw. die Moves, die die Heuristik bilden. Ein wesentliches Merkmal von Metaheuristiken ist die Akzeptanz schlechterer Lösungen im Laufe des Suchprozesses, um lokalen Extrema zu entkommen. Einen guten Einstieg in die Thematik der Metaheuristiken bieten Blum und Roli (2003) oder Gendreau und Potvin (2005).

Im Folgenden werden die Konzepte der naturanalogen Verfahren SimA und GA sowie der künstlichen Verfahren der VNS und TS ausführlich vorgestellt, da diese die Grundlage der Implementierungen, die im Verlauf der Arbeit vorgestellt werden, bilden.

2.3.3.1 Simulated Annealing (SimA)

SimA ist eines der ersten Verfahren, das es explizit ermöglicht, ein lokales Extremum zu überwinden (Blum und Roli, 2003, S. 274). Das Konzept als solches stammt von Metropolis et al. (1953) und gewann

[14] Hier wird aus Gründen der Übersichtlichkeit auf die Nennung des 2-0-Cross-Exchanges und 1-0-Cross-Exchanges, die ebenfalls ihr Äquivalent in den anderen Operatoren finden, verzichtet.

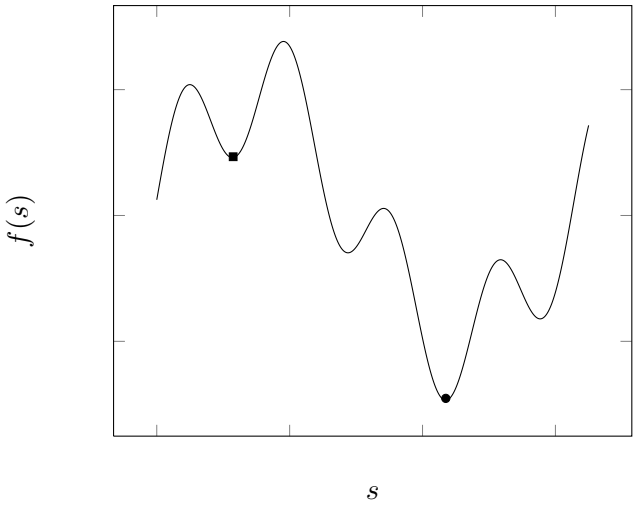


Abbildung 2.14: Schematische Darstellung eines Lösungsraumes

durch Kirkpatrick et al. (1983) an Popularität.[15] Die Grundidee ist angelehnt an den „process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuaration (i.e., its minimum lattice energy state), and thus is free of crystal defects. If the cooling schedule is sufficiently slow, the final configuaration results in an solid with such superior structural integrity“ (Nikolaev und Jacobson, 2010, S. 2). Hier zeigt sich die Analogie zur Natur, die dem Verfahren zugrunde liegt.

Diese Grundidee wird auf kombinatorische Optimierungsprobleme übertragen. In Algorithmus 1 ist der Verlauf des SimA schematisch dargestellt.

Algorithmus 1: Pseudocode des SimA in Anlehnung an Nikolaev und Jacobson (2010)

```
1  Wähle Initiallösung s ∈ S
2  Setze Initialtemperatur T
3  while Abbruchkriterium nicht erfüllt do
4  |  Setze m = 0
5  |  while m ≠ Anzahl der Iterationen auf Temperaturniveau do
6  |  |  Erzeuge zufällige Lösung s' ∈ N(s)
7  |  |  Berechne c_Δ = f(s') − f(s)
8  |  |  if c_Δ ≤ 0 then
9  |  |  |  s = s'
10 |  |  end
11 |  |  if c_Δ > 0 und rand(0,1) ≤ e^(−c_Δ/T) then
12 |  |  |  s = s'
13 |  |  end
14 |  |  m = m + 1
15 |  end
16 |  Aktualisierung von T
17 end
```

[15] Sowohl Kirkpatrick et al. (1983) als auch Černý (1985) haben zeitgleich das Konzept erarbeitet.

Dem Algorithmus wird eine Startlösung $s \in S$ übergeben. Im Falle des CVRPs können bspw. die Algorithmen, die in Abschnitt 2.3.2.1 vorgestellt wurden, die Startlösung erzeugen. Weiterhin wird die Starttemperatur initialisiert. Nun wird zufällig eine neue Lösung s' aus der Nachbarschaftsmenge $N(s) \subseteq S$ generiert bzw. gewählt. Wie sich die Nachbarschaft zusammensetzt, hängt vom Design des Verfahrens ab. So kann die Nachbarschaft über den Move bzw. den Operator definiert werden. Die Veränderung c_Δ ergibt sich aus dem neuen Zielfunktionswert $f(s')$ abzüglich des alten Zielfunktionswertes $f(s)$. Wenn $c_\Delta \leq 0$, so handelt es sich um eine Verbesserung der aktuellen Lösung[16], und die neu generierte Lösung ersetzt die zuvor aktuelle Lösung s. Sollte die neue Lösung jedoch einen schlechteren Zielfunktionswert aufweisen, wird zufällig eine gleichverteilte Zahl mit der Funktion $rand(0, 1)$ zwischen 0 und 1 erzeugt. Sollte diese Zahl kleiner oder gleich der Annahmewahrscheinlichkeit[17]

$$P_A = e^{-\frac{c_\Delta}{T}} \tag{2.30}$$

sein, wird die Lösung, obwohl sie die aktuelle verschlechtert, trotzdem als neue Lösung akzeptiert. Wenn die Zufallszahl nicht das Kriterium erfüllt, so ist die alte Lösung gleichzeitig die neue und der Suchprozess wird mit dieser weiter fortgesetzt. Die Temperatur T wird in Abhängigkeit der Anzahl der Iterationen, die auf einem bestimmten Temperaturniveau durchgeführt werden, angepasst.

Die Temperatur nimmt großen Einfluss auf die Ergebnisqualität und Laufzeit des Algorithmus. Je höher die Temperatur dabei ist, umso höher ist die Wahrscheinlichkeit, dass größere Verschlechterungen des Zielfunktionswertes akzeptiert werden (Blum und Roli, 2003, S. 274). So nimmt bereits die Wahl der Initialtemperatur entscheidenden Einfluss auf den Suchprozess. Eine zu hohe Temperatur führt dazu, dass fast jeder Move bzw. jede neue Lösung akzeptiert wird, d.h., die Suche springt in diesem Fall im Lösungsraum umher, ohne wirkliche Verbesserungen des Zielfunktionswertes zu erzielen. Wird sie zu niedrig gewählt, führt das dazu, dass kaum oder nur sehr wenige Verschlechterungen angenommen werden, wodurch die Suche sehr schnell in einem lokalen Extremum, das sie nicht überwinden kann, endet. Der Suchprozess bzw. die Temperaturen während des Suchprozesses sollten sich aus einer adäquaten Mischung von randomisierter und rein iterativer Suche zusammensetzen (Blum und Roli, 2003, S. 274).

Auf die Akzeptanz einer Lösung hat jedoch nicht nur die Temperatur Einfluss, sondern auch direkt der Zielfunktionswert. Sobald mit dem zufällig ausgewählten Move eine bessere Lösung generiert werden kann, wird diese angenommen. Andernfalls wird mit P_A geprüft, ob der Move trotz Verschlechterung akzeptiert wird. Die Annahmewahrscheinlichkeit wird vom Zielfunktionswert beeinflusst. Je größer die Verschlechterung der neuen Lösung im Vergleich zur vorherigen ist, umso geringer ist die Wahrscheinlichkeit, dass sie angenommen wird.[18] In Abbildung 2.15 ist der Verlauf der Annahmewahrscheinlichkeit P_A in Abhängigkeit von der Differenz des neuen und alten Zielfunktionswertes sowie der Temperatur T dargestellt. Es wird ersichtlich, dass bei konstantem Temperaturniveau, aber steigendem c_Δ, also einer größeren Verschlechterung, die Annahmewahrscheinlichkeit sinkt. Analog dazu steigt die Annahmewahrscheinlichkeit bei konstantem c_Δ mit steigender Temperatur.

Die vorangegangen Ausführungen zeigen, dass man mittels des Temperaturniveaus das Verhalten des

[16] Hier wird implizit angenommen, dass es sich um ein Minimierungsproblem handelt. Der SimA-Algorithmus lässt sich natürlich auch auf Maximierungsprobleme anwenden.

[17] P_A steht für P-Akzeptanz und wird in Anlehnung an Wendt (1995, S. 116) gewählt.

[18] Diesen Ausführungen liegt die Annahme zu Grunde, dass die Temperatur konstant gehalten wird, um den Effekt zu erkennen. Andernfalls kann man durch eine Erhöhung der Temperatur auch wieder eine höhere Annahmewahrscheinlichkeit erzielen.

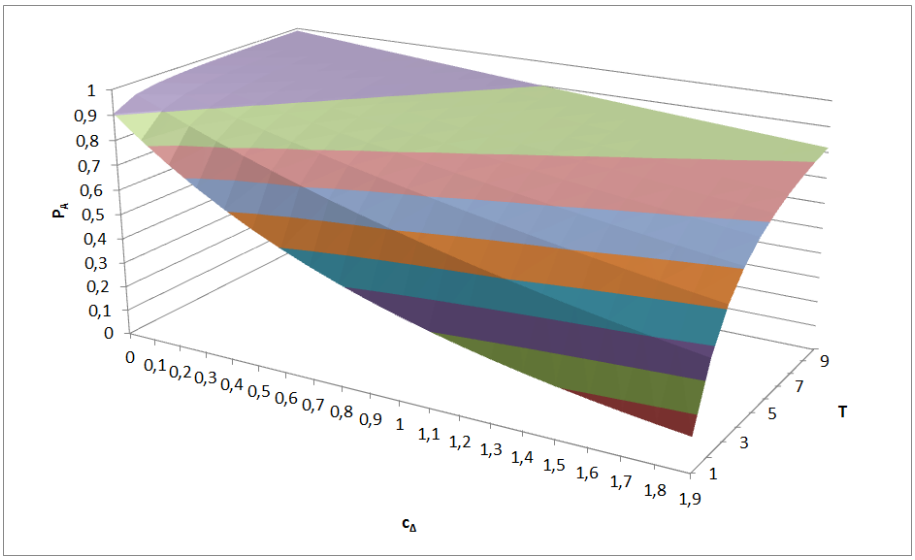


Abbildung 2.15: Annahmewahrscheinlichkeit in Abhängigkeit von c_Δ und T in Anlehnung an Wendt (1995, S. 116)

SimA stark beeinflussen kann.[19] Hierbei stellen sich vier grundlegende Fragen (Nikolaev und Jacobson, 2010, S. 25):

1. Wie hoch sollte die Initialtemperatur gewählt werden?
2. Wann sollte das Temperaturniveau jeweils angepasst werden?
3. Wie sollte das Temperaturniveau angepasst werden?
4. Bei welchem Temperaturniveau wird die Suche beendet?

Aus der Beantwortung der vier Fragen resultiert der sog. Temperaturschedule bzw. Abkühlungsplan. Eine Möglichkeit, die Initialtemperatur zu ermitteln, ist es, einige Iterationen des SimA-Verfahrens durchlaufen zu lassen und alle Moves, die dabei generiert werden, zu akzeptieren. Man berechnet den Mittelwert über alle c_Δ, die bestimmt wurden und legt dann die Temperatur so fest, dass Verschlechterungen auf dem Niveau des Mittelwertes mit einer vorgegebenen Wahrscheinlichkeit akzeptiert werden. Diese Idee stammt von Johnson et al. (1989, S. 870). Ein ähnliches Verfahren wird von Ben-Ameur (2004) vorgeschlagen. Auch hier werden einige „zufällige“ Lösungen gebildet, um darauf aufbauend die Initialtemperatur zu berechnen. Weiterhin gibt der Autor einen Überblick über viele weitere Verfahren, die zur Bestimmung der Starttemperatur veröffentlicht sind.

Die zweite und dritte Frage soll zusammenhängend betrachtet werden. Die Frage nach dem „Wann“ bedeutet nichts anderes, als die Frage wie lange neue Lösungen auf einem bestimmten Temperaturniveau generiert werden sollen, bevor ein anderes eingestellt wird. Bei der Frage nach dem „Wie“ kommt es darauf an, wie die Temperatur angepasst wird. Denkbare Temperaturanpassungen sind nach Schneider und Puchta (2010, S. 5823) bspw. logarithmische Abkühlungspläne

$$T_{new} = \frac{a}{b + log(t)} \tag{2.31}$$

[19] Man kann natürlich auch über die Zielfunktion bzw. über die Nachbarschaft Einfluss nehmen. Jedoch ist eine solche Änderung/Anpassung i.d.R. mit einem sehr hohen Aufwand verbunden.

wobei a und b problemspezifisch gewählt werden und t die bis dahin verstrichene Zeit ist, lineare Abkühlungspläne

$$T_{new} = T_{current} - \Delta T \tag{2.32}$$

wobei ΔT ein konstanter Wert ist, um den die Temperatur gesenkt wird, oder exponentielle Abkühlungspläne

$$T_{new} = \alpha \cdot T_{current} \tag{2.33}$$

wobei $0 < \alpha < 1$ der Kühlfaktor bzw. die Abkühlungsrate ist. An dieser Stelle sollte nur ein Eindruck der Gestaltungsmöglichkeiten, die aus dem Abkühlungsplan resultieren, gegeben werden. In den Beiträgen von Cohn und Fielding (1999), Varanelli und Cohoon (1999), Schneider und Puchta (2010) und Turky (2011) finden sich ausführliche Beschreibungen darüber, wie ein Abkühlungsplan erstellt werden kann. Vergleiche von verschiedenen Abkühlungsplänen werden in den Beiträgen von Nourani und Andresen (1998) und Suman und Kumar (2005) vorgenommen.

Die letzte Frage zielt darauf ab, wann der Algorithmus endet. Hier muss man sich fragen, ob das Verfahren nach einer bestimmten Zeit oder Iterationszahl abbricht oder wenn ein zuvor festgelegtes Temperaturniveau erreicht ist. Oftmals endet ein SimA-Verfahren dann, wenn die Temperatur auf 0 gesetzt werden müsste.[20] Allerdings ist dies nicht in allen Fällen praktikabel; insbesondere dann nicht, wenn ein Abkühlungsplan auch ein Wiederaufwärmen vorsieht.

Ein bedeutendes Merkmal des SimA ist, dass es - zumindest unter Einsatz eines bestimmten Abkühlungsplans - mit hoher Wahrscheinlichkeit gegen das globale Optimum strebt. Das Problem hierbei ist jedoch, dass ein sehr langsames Abkühlen notwendig ist, um die Optimallösung zu erreichen und es somit für viele Anwendungen aufgrund der Rechendauer nicht praktikabel ist (Blum und Roli, 2003, S. 275).

2.3.3.2 Tabusuche (TS)

Die TS gehört zur Klasse der künstlichen Verfahren, d.h. sie enthält keine Analogien zu Vorkommnissen in der Natur. Sie stammt von Glover (1986) basierend auf dem Beitrag von Glover (1977). Die Motivation von Glover (1986, S. 534) zur Einführung des Verfahrens besteht darin, dass Heuristiken oftmals in lokalen Optima enden, die sehr weit vom globalen Optimum entfernt sind. Er möchte mit der TS diese Problematik entschärfen.

Der Pseudocode der TS ist in Algorithmus 2 dargestellt.[21] Das Verfahren fordert ähnlich wie das SimA eine Initiallösung s. Diese Initiallösung wird direkt als beste bisher gefundene Lösung s^* übernommen.[22] Danach müssen noch ein Iterationszähler k und die Tabuliste TL initialisiert werden. Die Tabuliste enthält dabei die Moves, die aktuell nicht auf eine Lösung angewendet werden dürfen. Auf das Design der Tabuliste wird im weiteren Verlauf noch näher eingegangen. $MM(s)$ ist die Menge aller Moves, die

[20] Grundsätzlich ist es nur bei linearen Abkühlungsplänen möglich, eine Temperatur von 0 zu erreichen. Durch die beschränkte Genauigkeit heutiger Rechner kommt es jedoch auch bei anderen Abkühlungsplänen zu einem Erreichen von 0, da eine Rundung stattfindet.

[21] Auch hier wird im Pseudocode wieder ein Minimierungsproblem unterstellt. Allerdings ist die TS auch im Kontext eines Maximierungsproblems anwendbar.

[22] Das direkte Übernehmen der Initiallösung als beste Lösung setzt voraus, dass es sich dabei um eine gültige Lösung handelt. Sollte die TS jedoch auch ungültige Lösungen akzeptieren, so muss zunächst noch die Gültigkeit geprüft werden. Auf diese Gültigkeitsprüfung wird jedoch aufgrund der Übersichtlichkeit in der Darstellung verzichtet.

auf die Lösung s angewendet werden können. Mittels $MM(s) - TL \neq \emptyset$ wird überprüft, ob Moves auf der aktuellen Lösung durchgeführt werden dürfen; dabei werden von der Menge der möglichen Moves die Moves, die tabu sind, entfernt. Sollte daraus eine leere Menge resultieren, endet das Verfahren. Gleiches gilt für den Fall, wenn eine bestimmte Iterationszahl überschritten ist. Trifft beides nicht zu, so wird zunächst Zähler k um eins erhöht. Danach wird Move M aus der Menge der möglichen Moves, die nicht tabu sind, ausgewählt, unter der Bedingung, dass es keinen anderen Move gibt, der ein besseres Ergebnis auf der aktuellen Lösung erzielt. Mittels des so gewählten Moves M wird eine neue Lösung generiert, die als neue aktuelle Lösung fungiert. Ist der Zielfunktionswert der neuen Lösung besser als der der bisher besten gefundenen, so wird die neue Lösung als bisher beste gefundene gespeichert. Letztendlich wird die Tabuliste aktualisiert und die Iteration der TS abgeschlossen.

Algorithmus 2: Pseudocode TS in Anlehnung an Glover (1989)

```
1  Wähle Initiallösung s ∈ S
2  Setze s* = s
3  Setze Iterationszähler k = 0 und Tabuliste TL = ∅
4  while MM(s) − TL ≠ ∅ und bestimmte Iterationszahl nicht überschritten do
5  |  Setze k = k + 1
6  |  Wähle besten Move M ∈ MM(s) − TL
7  |  Erzeuge s durch Anwendung von M auf s
8  |  if f(s) < f(s*) then
9  |  |  Setze s* = s
10 |  end
11 |  Update TL
12 end
```

Grundsätzlich sucht die TS immer die beste Nachbarlösung, d.h., sie findet i.d.R. sehr schnell ein lokales Optimum. Das Verfahren erlaubt jedoch nicht nur verbessernde Moves, sondern auch verschlechternde. Wenn die TS also in einem lokalen Optimum angelangt ist, sucht der Algorithmus nach dem besten Move - oder anders ausgedrückt - nach der besten bzw. geringsten Verschlechterung. Danach gilt diese gefundene Lösung als Ausgangspunkt für die weitere Suche. Nun wird wieder die beste Nachbarlösung gesucht. An dieser Stelle kommt die Tabuliste zum Einsatz. Die beste Nachbarlösung ist eigentlich die Lösung im lokalen Optimum, die man gerade erst verlassen hat. Man würde also wieder zurück zum lokalen Optimum springen und es entstünde ein Zyklus. Die Zyklusbildung wird jedoch von der Tabuliste unterbunden, da eine bestimmte Anzahl von zuletzt ausgeführten Moves tabu gesetzt ist (Gendreau und Potvin, 2010, S. 46). Somit ist es dem Verfahren nicht erlaubt, zurück zum lokalen Extremum zu gelangen und es kommt nicht zu einem Zyklus.

Es wird deutlich, dass die Tabuliste ein wichtiger Gestaltungsparameter für den Erfolg einer TS ist. Es existiert keine einhellige Meinung darüber, was genau eine Tabuliste speichern soll. In der Ursprungsarbeit von Glover (1986) wird vorgeschlagen, dass einzelne Moves tabu gesetzt werden bzw. in der Tabuliste gespeichert werden. Dennoch gibt es viele weitere Verfahren, um die Tabuliste zu füllen. So ist es möglich, ganze Lösungen zu speichern und diese Lösungen tabu zu setzen oder aber auch bestimmte Merkmale, die eine Lösung enthalten kann, tabu zu setzen. Ersteres wird vergleichsweise

selten genutzt, da einerseits der damit einhergehende Speicheraufwand immens ist und andererseits die Prüfung, ob eine ganze Lösung tabu ist, recht teuer ist (Gendreau und Potvin, 2010, S. 46). Auf die Verwaltung der Tabuliste an sich, soll an dieser Stelle nicht näher eingegangen werden, sondern es wird auf den Beitrag von Glover (1990) verwiesen, wo sich einige Managementverfahren finden. Die Beschreibung der Umsetzung der Tabuliste in dieser Arbeit wird in Abschnitt 3.1.4 erläutert.

Eine weitere Schlüsselkomponente des Verfahrens, die ebenfalls mit der Tabuliste zusammenhängt, ist die Länge bzw. Dauer des Tabustatus eines Moves. Es muss festgelegt werden, wie lange ein Move nicht angewendet werden darf bzw. wie viele Elemente in der Tabuliste aufgenommen werden können. Auch hierbei existieren verschiedene Ansätze. So kann beim Design des Verfahrens eine statische Länge der Tabuliste festgelegt werden oder es wird bspw. bei jedem Move, der der Tabuliste hinzugefügt wird, eine Zufallszahl generiert, die angibt wie lange er tabu ist. Je nach Verfahren, das die Länge der Tabuliste bestimmt, wird dadurch auch implizit das Management der Tabuliste vorgegeben (Gendreau und Potvin, 2010, S. 47). Falls die Länge der Tabuliste zu groß gewählt ist, kann es im Extremfall dazu führen, dass alle Moves, die zu anderen Lösungen führen, tabu sind und somit die Suche abgebrochen wird (vgl. Algorithmus 2). Ist die Listenlänge hingegen zu kurz gewählt, besteht die Gefahr, dass es im Suchprozess zu Zyklen kommt. Wenn man von den Extremfällen einmal absieht, so kann man über die Länge der Tabuliste bestimmen, wie groß der Bereich des Lösungsraumes ist, der von der TS abgesucht wird. Bei kleiner Größe wird auch nur ein kleiner Bereich des Suchraumes betrachtet werden, wohingegen bei größeren Werten ein großer Bereich des Suchraums betrachtet wird (Blum und Roli, 2003, S. 276). Eine allgemeine Aussage bzgl. der Länge ist jedoch schwer zu treffen und problemspezifisch.[23]

Eine Komponente, die in den meisten Implementierungen von TS enthalten ist, ist das Aspirationskriterium. Angenommen man hat eine entsprechend lange Tabuliste gewählt und befindet sich nun an einem Punkt im Lösungsraum von dem aus man mit einem Move, der aktuell tabu ist, eine neue beste Lösung, d.h. ein neues bestes lokales Optimum, finden kann, so verbietet die Tabuliste die Ausführung des Moves. Um diesen Nachteil zu umgehen, wird das Aspirationskriterium verwendet, das definiert, wann ein Move - trotz Tabustatus - ausgeführt werden darf. Oftmals wird als Kriterium die Zielfunktion herangezogen, d.h. falls durch einen Move, der tabu ist, der Zielfunktionswert besser ist als der bisher beste gefundene, wird der Move trotzdem ausgeführt und man gelangt zu einer neuen besten Lösung (Gendreau und Potvin, 2010, S. 47).

Die TS baut genauso wie das SimA im Rahmen des CVRP zumeist auf lokalen Suchoperatoren auf. Es wird in beiden Algorithmen eine Nachbarlösung aus der Menge $N(s)$ gesucht mit dem Unterschied, dass bestimmte Lösungen aus $N(s)$ mittels der TS nicht erreicht werden können, da die Moves, die zu diesen Lösungen führen, in der Tabuliste enthalten sind und nicht ausgeführt werden dürfen. Demgegenüber kann der SimA-Algorithmus theoretisch alle Lösungen erreichen. Unterschiede ergeben sich in der Wahl der Nachbarlösung. Wo seitens der TS versucht wird, den besten Move bzw. Lösung aus der Nachbarschaft zu wählen, wird beim SimA ein zufälliger Move gewählt und geprüft, ob er zur Transition der alten in die neue Lösung akzeptiert wird. Ein weiterer fundamentaler Unterschied ergibt sich bei der Bestimmung der besten Lösung. Wie in Algorithmus 1 ersichtlich, wird bei Anwendung des SimA

[23] Auch wenn Glover (1986, S. 543) in seiner Arbeit ausführt, dass eine gute Wahl für die Länge der Tabuliste 7 sein solle - unabhängig vom Anwendungsgebiet - sollte man sich nicht dazu verleiten lassen, eine statische Länge von 7 im eigenen Anwendungsbereich zu verwenden. Die Aussage kann jedoch als Ausgangsbasis für Tests herangezogen werden, um eine adäquate Größe der Tabuliste für den eigenen Anwendungsfall zu bestimmen.

die beste Lösung, die im Verlauf des Suchprozesses gefunden wird, nicht gespeichert. Der Rückgabewert, den das Verfahren liefert, ist somit immer die letzte aktuelle Lösung. Dabei ist es durchaus denkbar, dass diese schlechter als die beste gefundene im gesamten Suchprozess ist. Wenn man jedoch als Abbruchkriterium eine sehr niedrige Temperatur wählt, so ist durch das Konvergenzverhalten der Methode die Wahrscheinlichkeit einer großen Abweichung zur bisher besten gefundenen Lösung sehr gering. Demgegenüber steht die TS, bei der zu jeder Zeit die beste Lösung gespeichert wird. Somit wird garantiert, dass der Rückgabewert die tatsächlich beste gefundene Lösung des gesamten Suchprozesses ist. Man könnte an dieser Stelle mit dem höheren Speicheraufwand, der mit der TS verbunden ist, argumentieren. Allerdings dürfte das zu heutiger Zeit bzw. bei Verwendung aktueller Hardware und entsprechendem Speicherausbau keine Rolle mehr spielen. Insofern bietet sich auch eine Diskussion darüber an, inwiefern es sinnvoll sein könnte, die beste gefundene Lösung im Suchprozess des SimA zu speichern.

2.3.3.3 Variable Neighborhood Search (VNS)

Die VNS wird in dem Beitrag von Mladenović und Hansen (1997) eingeführt. Es handelt sich dabei wie die TS um ein künstliches Verfahren. Die Grundidee ist, dass die Nachbarschaften, aus denen neue Lösungen gewonnen werden, dynamisch angepasst werden (Hansen et al., 2010, S. 62). Durch die dynamische Anpassung sollen große Bereiche des Suchraums erreicht werden.

Algorithmus 3 enthält den Pseudocode der VNS. Zunächst muss festgelegt werden, welche Nachbarschaften im Verlauf des Verfahrens zur Anwendung kommen. Auch dieser Algorithmus verlangt, wie das SimA und die TS, eine Initiallösung und eine Abbruchbedingung. Danach beginnen die eigentlichen Iterationen. Zunächst wird eine zufällige Lösung s' aus der Nachbarschaft $N_M(s)$ erzeugt. M gibt dabei die aktuelle Nachbarschaft[24] an, die verwendet werden soll, um die Lösung zu generieren. Danach wird diese mittels einer Lokalsuche optimiert, d.h. bis das lokale Optimum, ausgehend von der zufälligen Lösung, gefunden ist. Es wird dann geprüft, ob dieses lokale Optimum s'' einen besseren Zielfunktionswert besitzt als das bisherige beste.[25] Sollte das der Fall sein, so wird die bisher beste gefundene Lösung durch das lokale Optimum ersetzt und die Nachbarschaft, die im nächsten Shaking-Schritt zur Anwendung kommt, wird auf 1 gesetzt. Liegt keine Verbesserung vor, wird im nächsten Shaking-Schritt eine andere Nachbarschaft verwendet. Das wird so lange wiederholt, bis alle Nachbarschaften zum Shaking herangezogen wurden und es in keiner dieser Nachbarschaften mehr zu einer Verbesserung der besten Lösung kommt. In diesem Fall wird, bis das Abbruchkriterium erfüllt ist, die Nachbarschaft, die zum Shaking genutzt wird, wieder auf 1 gesetzt. Im Wesentlichen besteht die VNS aus den Methoden Shaking, Lokalsuche und Ausführung des Moves (Blum und Roli, 2003, S. 279), die im Folgenden näher betrachtet werden.

Das Shaking sorgt dafür, dass neue zufällige Lösungen, ausgehend von der aktuellen Lösung, erzeugt werden. Die zufällige Generierung ist wichtig, damit der Suchprozess nicht nur einen zu kleinen Teil des Lösungsraums betrachtet. Das heißt, jedes Mal wenn die Shaking-Prozedur aufgerufen wird, wird die Suche zu einer anderen Stelle im Lösungsraum gelenkt, von der aus dann wieder ein lokales Minimum

[24] Die Nachbarschaft wird hier sozusagen durch den Move M determiniert.

[25] Im Pseudocode wird wieder angenommen, dass es sich um ein Minimierungsproblem handelt. Das ändert jedoch nichts an der Allgemeingültigkeit des Verfahrens bzw. der Möglichkeit, dass auch Maximierungsprobleme damit bearbeitet werden können.

Algorithmus 3: Pseudocode des VNS-Algorithmus in Anlehnung an Hansen und Mladenović (2001)

```
1  Wähle Nachbarschaftsstrukturen N_M, M = 1, ..., M_max
2  Setze Initiallösung s ∈ S
3  Wähle Abbruchbedingung
4  while Abbruchbedingung nicht erfüllt do
5  |  M = 1
6  |  while M ≤ M_max do
7  |  |  Shaking: Erzeuge eine zufällig Lösung s' ∈ N_M(s)
8  |  |  s'' = Lokalsuche(s') (lokales Optimum)
9  |  |  if f(s'') < f(s) then
10 |  |  |  s = s''
11 |  |  |  M = 1
12 |  |  else
13 |  |  |  M = M + 1
14 |  |  end
15 |  end
16 end
```

bestimmt wird. Man kann mit Blum und Roli (2003, S. 279) übereinstimmen, dass es nicht sinnvoll ist, sich durch das Shaking-Verfahren „zu weit“ von der aktuellen Lösung wegzubewegen, da sonst die VNS nichts anderes als ein Random-Multi-Start-Verfahren wäre. Das Ziel des Shakings sollte demnach sein, einen Punkt im Lösungsraum zu finden, von dem aus man mittels der Lokalsuche in ein anderes lokales Optimum als das derzeitige gelangt.

Der größte Einflussfaktor auf die Bestimmung des Nachbarn durch das Shaking sind die verwendeten Nachbarschaftsstrukturen N_M. Die Nachbarschaftsstruktur legt dabei die Menge der Moves fest, aus denen zufällig einer ausgewählt wird, um so die aktuelle Lösung in eine zufällige neue zu transformieren. Beim Design einer VNS steht man so vor der Entscheidung, welche und wie viele verschiedene Nachbarschaftsstrukturen verwendet werden sollen. Im Extremfall ist es denkbar, dass nur eine Nachbarschaft zum Einsatz kommt; in der Regel werden jedoch mehrere eingesetzt und die Nachbarschaften haben die Eigenschaft, dass $N_1 \subset N_2 \subset ... \subset N_{M_{max}}$ (Blum und Roli, 2003, S. 279), d.h., die Nachbarschaften sind der Größe nach geordnet. Hintergrund dessen ist, dass man die neue Lösung zunächst nur ein kleines Stück von der aktuellen Lösung entfernen möchte. Führt dies zu keinem Erfolg und es wird keine neue beste Lösung durch die kleine Nachbarschaft gefunden, wird die nächst größere verwendet usw. (vgl. Abbildung 2.16).[26]

Nachdem die zufällige Lösung erzeugt ist, gilt es von dieser aus eine Lokalsuche zu starten. Die Lokalsuche kann dabei eine oder mehrere der Nachbarschaftsstrukturen verwenden, die auch schon im Shaking-Verfahren zum Einsatz kommen. Dies ist jedoch nicht obligatorisch; es ist genauso möglich, dass in der Lokalsuche eine vollständig andere Nachbarschaft zum Einsatz kommt (Mladenović und Hansen,

[26] Ein treffendes Beispiel für solche wachsende Nachbarschaftsstrukturen findet man bei der Betrachtung des TSP. So könnte man hier für N_1 den 2-Opt, für N_2 den 3-Opt und für N_M den $(M+1)$-Opt wählen. Für diesen Fall würde $N_1 \subset N_2 \subset ... \subset N_M$ gelten.

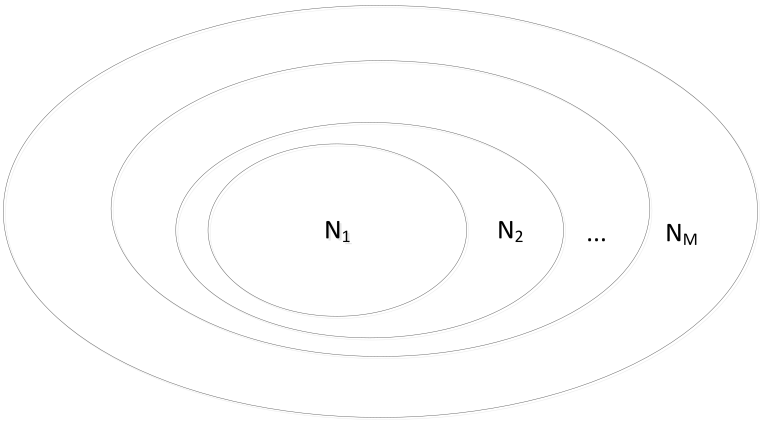


Abbildung 2.16: Nachbarschaftsrelation der VNS in Anlehnung an Pisinger und Ropke (2007, S. 2409)

1997, S. 1098). In Übereinstimmung mit Mladenović und Hansen (1997, S.1098) bietet es sich oft an, eine bzw. mehrere Nachbarschaften, die auch im Shaking-Verfahren verwendet werden, zu nutzen, da die entsprechende Implementierung schon vorliegt und die Entwicklung einer VNS dadurch vereinfacht und beschleunigt wird. Allerdings liegt die Anzahl der Nachbarschaftsstrukturen, die in der Lokalsuche zum Einsatz kommen, meist im Bereich zwischen eins und zwei (Hansen und Mladenović (2001, S. 452), Gendreau und Potvin (2010, S. 63)). Es ist aber ebenso möglich, eine weitere Metaheuristik in die VNS an Stelle der Lokalsuche (vgl. Hansen und Mladenović (2003, S. 150)), die nur auf wenigen Nachbarschaftsstrukturen basiert, zu verwenden, die dann ihrerseits ebenfalls zu einem lokalen Optimum führt. Auch wenn zuvor davon gesprochen wurde, dass die Lokalsuche, die bei einer VNS zum Einsatz kommt, in ein lokales Optimum führt, ist das keine notwendige Bedingung für eine Lokalsuche, die verwendet wird.

Im letzten Schritt muss entschieden werden, ob die ursprüngliche durch die neu generierte Lösung ersetzt wird. Das geschieht im einfachsten Fall nur dann, wenn der Zielfunktionswert der neuen Lösung besser ist als der der alten. Die VNS liefert in Analogie zur TS und im Gegensatz zum SimA die beste Lösung, die über den gesamten Suchprozess hinweg gefunden wurde.

Im Vergleich zur TS hat die VNS den Vorteil, dass man sich bei der Implementierung nicht um eine Tabuliste und die Bestimmung der Parameter, d.h. die Länge der Tabuliste kümmern muss. Anderseits ist es für eine erfolgreiche VNS unabdingbar, mehrere Nachbarschaften zu entwickeln, die im Shaking-Verfahren genutzt werden. Außerdem muss noch festgelegt werden, in welcher Reihenfolge diese zum Zuge kommen. Das dürfte die Vorteile gegenüber der TS, die i.d.R. auch mit nur einer Nachbarschaftsstruktur gute Ergebnisse erzielen kann, wieder aufwiegen. Gerade die Nachbarschaftsstruktur ist stark von der zu lösenden Problemstellung abhängig, wodurch bei einer Veränderung der Problemstruktur alle Strukturen entsprechend angepasst werden müssen. Bei der TS müssen demgegenüber lediglich die Tabuliste und wenige Nachbarschaftsstrukturen verändert werden.

2.3.3.4 Genetischer Algorithmus (GA)

GAs haben mit den anderen in dieser Arbeit betrachteten Algorithmen am wenigsten gemein. Ein Meilenstein in der Entwicklung der GA wird durch die Arbeit von Holland (1975) gelegt. Allerdings beschäftigen sich auch bereits Jahre vor dieser Entwicklung Autoren mit der Thematik (vgl. z.B. Rechenberg (1973)). GAs versuchen das Paradigma „survival of the fittest“, das in der natürlichen

Evolution vorkommt, zu imitieren (Blum und Roli, 2003, S. 285).

Algorithmus 4: Pseudocode eines GA in Anlehnung an Reeves (2010)

```
1  Wähle eine initiale Population
2  while Abbruchbedingung nicht erfüllt do
3  |  k = 0
4  |  while k < k_max do
5  |  |  if Crossover-Bedingung erfüllt then
6  |  |  |  Wähle Eltern
7  |  |  |  Führe Crossover aus
8  |  |  end
9  |  |  if Mutations-Bedingung erfüllt then
10 |  |  |  Führe Mutation aus
11 |  |  end
12 |  |  Evaluiere die Nachkommen
13 |  |  Update k
14 |  end
15 |  Wähle neue Population
16 end
```

In Algorithmus 4 ist ein vereinfachter GA dargestellt. Da es z.T. sehr starke Unterschiede zu den anderen hier vorgestellten Konzepten gibt, soll zunächst auf die Grundbestandteile, aus denen sich das Verfahren zusammensetzt, eingegangen werden. Zunächst sei die Population genannt. Dabei handelt es sich um eine Menge an Lösungen, d.h. in GAs wird nicht wie in Verfahren, die auf Nachbarschaftsstrukturen aufbauen, mit nur einer Lösung gearbeitet, die schrittweise verbessert wird, sondern eine Menge an Lösungen bildet den Ausgangspunkt. Diese werden auch als Individuen oder Chromosomen bezeichnet (Reeves, 2010, S. 112). Die Chromosomen werden in einen Reproduktionsprozess geschickt. Eine Selektionsfunktion bestimmt, welche Individuen im Reproduktionsprozess verwendet werden und damit die Grundlage für den Crossover bilden. Nachdem der Crossover durchgeführt ist, werden die reproduzierten Individuen mit einer bestimmten Wahrscheinlichkeit einer Mutation ausgesetzt. Schließlich werden die Nachkommen evaluiert und die Anzahl der Nachkommen k wird aktualisiert. Dieser Prozess wird so lange wiederholt, bis eine Mindestanzahl k_{max} an Nachkommen erreicht ist. Im letzten Schritt gilt es, die Population zu aktualisieren.

Auch der GA erwartet wie die anderen vorgestellten Metaheuristiken Initiallösungen. Es sollten grundsätzlich mehrere Individuen in einer Population enthalten sein, wobei darauf zu achten ist, dass keine bzw. nur sehr wenige identische Individuen enthalten sind. Bei der Initialisierung eines GA stellt

$$
\begin{array}{llll}
E_1\colon & 1\,0\,1\,1\,1\,0\,1\,1 & K_1\colon & 1\,0\,1\,1\,0\,1\,0\,0 \\
 & \mathrm{X} & \rightarrow & \\
E_2\colon & 1\,0\,0\,1\,0\,1\,0\,0 & K_2\colon & 1\,0\,0\,1\,1\,0\,1\,1
\end{array}
$$

Abbildung 2.17: Beispielhafter One-Point-Crossover in Anlehnung an Reeves (2010)

sich an das Konstruktionsverfahren die Anforderung, möglichst unterschiedliche Lösungen zu erzeugen.[27]

Der erste Schritt in einer Iteration des GA besteht darin, die Individuen, die zur Reproduktion verwendet werden sollen, zu selektieren. Dazu ist es in den meisten Anwendungsfällen unerlässlich, die Fitness bzw. den Zielfunktionswert, den eine Lösung besitzt, zu kennen und mit in die Selektionsphase einfließen zu lassen. Sie „bildet die Grundlage der Berechnung eines Überlebens- oder Selektionswertes für jedes Individuum“ (Wendt, 1995, S. 52). Das Design der Selektion erfolgt zumeist mit dem Ziel, dass die besseren Individuen - also die mit dem besseren Zielfunktionswert - auch häufiger zur Reproduktion eingesetzt werden. Erreicht werden kann das durch die sog. Roulette-Wheel-Selektion (Reeves, 2010, S. 120f.). Dabei steigt die Wahrscheinlichkeit eines Individuums zur Reproduktion ausgewählt zu werden mit besserem Zielfunktionswert. Gleichzeitig werden dadurch jedoch vermeintlich schlechte Individuen nicht vollständig ausgeschlossen, da es immer noch eine gewisse Wahrscheinlichkeit gibt, dass auch sie zur Reproduktion herangezogen werden. Das sorgt vor allem dafür, dass eine gewisse Diversifität in der Population erhalten bleibt.

Nachdem die Selektion abgeschlossen ist, gilt es, die Reproduktion bzw. den Crossover mit Hilfe der Elternlösungen durchzuführen. Es gibt unterschiedlichste Arten von Crossovern, wobei die meisten gemein haben, dass aus zwei Elternlösungen eine oder mehrere Nachwuchslösungen erzeugt werden.[28] Je nach Art des Crossovers spielt die Repräsentation der Chromosomen eine entscheidende Rolle. Die einfachste Darstellung von Chromosomen ist die der Binärrepräsentation. Dabei wird ein Individuum mit Nullen und Einsen codiert. Ein möglicher Crossover in dieser Repräsentation ist der One-Point-Crossover, der in einem Beispiel in Anlehnung an Reeves (2010, S. 113) verdeutlicht werden soll. Gegeben sind zwei Elternteile E_1 und E_2 (vgl. Abbildung 2.17), die zur Anwendung eines One-Point-Crossovers an der Stelle 4 in den Chromosomen herangezogen werden. Zunächst stellt E_1 die Basis für den Reproduktionsprozess dar und es werden die ersten 4 Stellen des Chromosoms von E_1 sowie die Stellen ab Position 4 des Chromosoms E_2 zu einem neuen Individuum K_1 zusammengefügt. In Analogie dazu wird K_2 erzeugt, nur dass nun E_2 die Basis darstellt. Damit werden aus den Elternteilen zwei neue Nachkommen erzeugt, die den weiteren Prozess des GA durchlaufen können. Es gibt unzählige verschiedene Crossover-Arten auf die an dieser Stelle nicht weiter eingegangen werden soll. Allerdings sei noch angemerkt, dass je nach zu lösendem Problem und seiner Repräsentation nicht jeder Crossover angewendet werden kann.

Sobald die Generierung des Nachwuchses abgeschlossen ist, wird zufällig bestimmt, ob ein neu

[27] Um zu verdeutlichen wie unterschiedliche Lösungen generiert werden können, soll das Sweep-Verfahren herangezogen werden. Hier kann man je nach Wahl des Ausgangswinkels einfach unterschiedliche Lösungen generieren, indem man den Ausgangswinkel für jedes neu hinzuzufügende Individuum zufällig wählt. Eine andere Möglichkeit besteht darin, vollständig randomisierte Ausgangslösungen zu generieren, wobei das den weiteren Suchprozess verlangsamen kann, d.h., es dauert eine längere Zeit bis gute Lösungen gefunden werden. Allerdings wird genau dieses Vorgehen von den meisten Autoren befürwortet, um nicht schon zu Beginn die Population in eine bestimmte Richtung zu lenken (Reeves, 2010, S. 119).

[28] Beiträge, die sich damit beschäftigen, dass mehr als zwei Eltern an einem Crossover beteiligt sind, stammen bspw. von Eiben et al. (1994) und Tsutsui und Ghosh (1998).

erzeugtes Individuum einer Mutation unterzogen wird. Dazu wird ein Mutationsoperator verwendet. Ein Beispiel für einen solchen ist für die binäre Chromosomen-Repräsentation das Komplement, das mit einer bestimmten Wahrscheinlichkeit eine 0 in 1 wandelt und umgekehrt (Wendt, 1995, S. 58). Das Ziel der Mutation besteht darin zu verhindern, dass eine Population zu schnell konvergiert (Blum und Roli, 2003, S. 286). In diesem Fall ähneln sich Individuen sehr, und es können somit keine anderen Bereiche des Suchraums mehr erreicht werden. Die Mutation kann man also als eine Diversifizierungsmethode ansehen.

Im letzten Schritt einer GA-Iteration gilt es zu bestimmen, was mit den erzeugten Individuen geschieht. Es stellt sich demnach die Frage, ob bzw. wie sie Teil der Population werden. Auch hier kommt es also wieder zu einer Selektion. Außerdem muss an dieser Stelle entschieden werden, was mit den alten Individuen der Population geschieht. Ein denkbarer Ansatz ist es bspw., so viele Nachkommen wie Individuen, die in der Initialpopulation enthalten sind, zu erzeugen und sämtliche alte gegen die neu erzeugten Individuen zu ersetzen. Das birgt jedoch die Gefahr, dass sehr gute Chromosomen komplett gelöscht werden und nicht mehr zur Reproduktion zur Verfügung stehen. Deshalb wird nur selten die gesamte Population ausgetauscht, sondern es wird vielmals versucht, die besten Individuen in der Population zu erhalten (Hertz und Kobler, 2000, S. 4).

Ein großes Problem, das bei Anwendung eines GA im Rahmen des CVRP entsteht, ist die Gewährleistung der Gültigkeit von Lösungen. Angenommen, man hat in Analogie zu dem Beispiel in Abbildung 2.17 zwei Elternlösungen als Informationsspender. Die Chromosomen sind dabei so aufgebaut, dass 0 das Depot repräsentiert, die Zahlen zwischen 1-5 die Kunden und die jeweilige Reihenfolge wie sie auf der Route bedient werden. Die 0 dient als Separator für die verschiedenen Touren (vgl. Abbildung 2.18).[29] Wie man leicht erkennen kann, sind beide Lösungen, die der Crossover erzeugt, ungültig. Es werden einerseits nicht mehr alle Kunden bedient und andererseits werden manche Kunden doppelt beliefert. Unter diesen Umständen, gibt es nach Hertz und Kobler (2000, S. 5) drei mögliche Vorgehensweisen. Eine der einfachsten Varianten ist das Verbieten von ungültigen Lösungen, d.h. sobald der Crossover eine solche generiert, wird sie direkt wieder verworfen. Eine andere Möglichkeit ist es, eine Strafkostenfunktion zu nutzen, d.h. die Wahrscheinlichkeit, dass ungültige Individuen in die neue Population bzw. zur Vermehrung herangezogen werden, sinkt rapide. Zuletzt kann man Reparaturfunktionen, die dem Crossover und der Mutation nachgelagert sind und dafür sorgen, dass die ungültigen Chromosomen wieder zu gültigen werden, verwenden. Ein anderer Ansatz zum Vermeiden ungültiger Lösungen ist es, entsprechende, also problemadäquate Crossover- und Mutations-Verfahren zu verwenden oder die Problemrepräsentation an die Crossover- und Mutations-Methoden anzupassen. In der Arbeit von Wendt (1995, S. 81ff.) finden sich einige Beispiele für Crossover-Operatoren, die die Pfadrepräsentation nutzen.[30] Auch im Rahmen des CVRPs kommt durch die Repräsentation der Lösungen als große Tour eine Pfadrepräsentation zum Einsatz, weshalb sie von Bedeutung für diese Arbeit ist. Die Implementierung dieser Arbeit verwendet einen angepassten Order-Crossover (OX) (vgl. Abschnitt 3.4.2).

Wie bei den anderen in dieser Arbeit vorgestellten Metaheuristiken sind auch bei einem GA viele

[29] Auf die Datenstrukturen, die zur Repräsentation des CVRPs verwendet werden können, wird in Kapitel 3.1.1 detaillierter eingegangen.

[30] Auch wenn in der Arbeit von Wendt (1995) von einem TSP gesprochen wird, so halten sich die Unterschiede zu einem CVRP wie sie an dieser Stelle genutzt werden in Grenzen.

$$
\begin{array}{llll}
E_1\text{:} & 0\,1\,3\,2\,0\,5\,4\,0 & K_1\text{:} & 0\,1\,3\,2\,3\,1\,4\,0 \\
 & \mathrm{X} & \rightarrow & \\
E_2\text{:} & 0\,5\,2\,0\,3\,1\,4\,0 & K_2\text{:} & 0\,5\,2\,0\,0\,5\,4\,0
\end{array}
$$

Abbildung 2.18: Beispielhafter One-Point-Crossover im Kontext des CVRPs

Parameter, die für den Erfolg des Verfahrens verantwortlich sind, zu bestimmen. Der GA hebt sich von den anderen am stärksten ab, da er mit mehreren Lösungen operiert, wohingegen die anderen i.d.R. nur auf einer Lösung arbeiten und diese iterativ verbessern. Der GA „springt“ hingegen sehr viel mehr im Lösungsraum umher. Ein großer Vorteil, den ein GA gegenüber anderen Verfahren mitbringt, ist sein intrinsischer Parallelismus (Holland, 1975, S. 71). Bei SimA, TS und VNS handelt es sich grundsätzlich um einen rein iterativen Prozess, der sich nicht selbstverständlich parallelisieren lässt. Neben der Gemeinsamkeit, dass sowohl SimA als auch GA zu den naturanalogen Verfahren gehören, zeigt sich, dass auch der GA nicht zwangsläufig die beste gefundene Lösung, die während des gesamten Suchprozesses ermittelt wird, liefert. Das hängt sehr stark davon ab, wie die neue Population gebildet wird und was mit den besten Lösungen geschieht.

2.4 Parallelisierung

Ein wesentlicher Grund, warum in den vergangenen Jahren verstärkt Multiprozessor-Systeme und somit Parallelisierung an Bedeutung gewinnen[31], liegt darin begründet, dass durch die Steigerung der Taktrate der Energieverbrauch sehr viel stärker zunimmt (Blake et al., 2009, S. 27). Somit wären Leistungssteigerungen nicht mehr bzw. nur unter erheblichem Mehraufwand durch alleinige Steigerung der Taktrate möglich gewesen. Weiterhin wirkt nach Rauber und Rünger (2012, S. 3) „die Endlichkeit der Übertragungsgeschwindigkeit der Signale als limitierender Faktor“.[32]

In diesem Abschnitt soll allgemein auf Parallelisierungstechniken eingegangen werden. Hardwarespezifische Besonderheiten werden bewusst ausgeklammert und erst in Kapitel 2.7 zur Beschreibung der verwendeten Hardwarearchitektur in dieser Arbeit näher betrachtet. Zunächst werden Charakterisierungsmöglichkeiten für parallele Algorithmen vorgestellt, bevor dann auf den Vergleich von sequentiellen mit parallelen Programmen eingegangen wird.

[31] Parallele Rechnersysteme gibt es schon viele Jahre. Die Aussage soll jedoch ausdrücken, „dass durch die künftige Allgegenwärtigkeit der Multicore-Prozessoren ein Ausbreiten paralleler Programmiertechniken in alle Bereiche der Softwareentwicklung erwartet“ (Rauber und Rünger, 2008, S. 21) werden kann.

[32] Zur Verdeutlichung dieses Zusammenhangs ziehen Rauber und Rünger (2012, S. 3-4) folgendes Beispiel heran: „Ein mit einer Taktrate von 3 GHz arbeitender Prozessor hat entsprechend eine Zykluszeit, also eine Dauer eines Taktes, von etwa $0,33ns$. In dieser Zeit kann ein Signal eine Entfernung von $0,33 \cdot 10^{-9}s \cdot 0,3 \cdot 10^{9}\frac{m}{s} \approx 10cm$ zurücklegen, wobei als Obergrenze der Übertragungsgeschwindigkeit die Lichtgeschwindigkeit im Vakuum $(0,3 \cdot 10^{9}\frac{m}{s})$ angenommen wird. Bei einer Verzehnfachung der Taktrate würde die Zykluszeit entsprechend zehnmal kleiner und die Signale könnten in einem Zyklus also gerade noch $1cm$ zurücklegen, womit die Größenordnung der Ausdehnung eines Prozessorchips, die aktuell zwischen 200 und $400mm^2$ liegt, erreicht wäre. Der Transfer von Signalen zwischen zwei beliebigen Positionen auf dem Prozessorchip könnte also nicht innerhalb eines Taktes durchgeführt werden. Die maximal nutzbare Taktrate wird damit von den Signallaufzeiten bestimmt.“

2.4.1 Charakterisierung paralleler Algorithmen

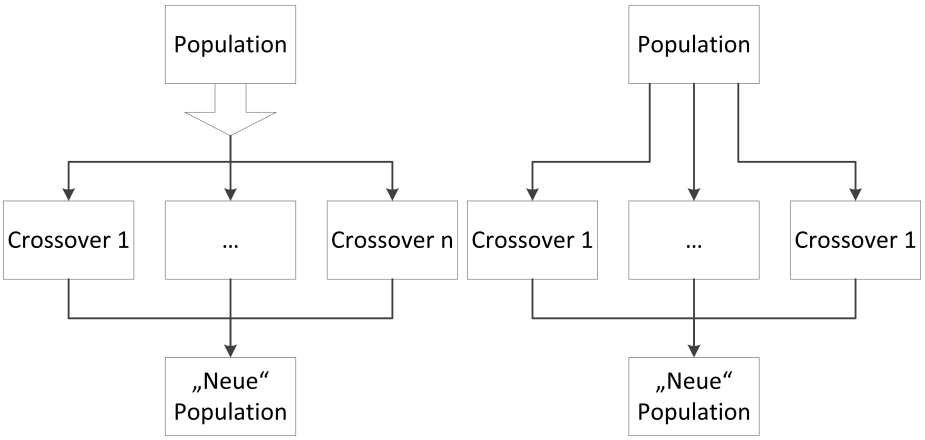
Das Ziel, das bei der Parallelisierung im Vordergrund steht, ist, dass die Zeit, die ein Programm zur Ausführung benötigt, verkürzt wird (Rauber und Rünger, 2012, S. 4). Um das zu erreichen, ist es nötig, alle zur Verfügung stehenden Mittel - also Rechnerkapazitäten - zu nutzen und nicht, wie es in vielen Anwendungen der Fall ist, nur einen Rechnerkern zu nutzen und die anderen Kerne, die ebenfalls zur Verfügung stehen, während des Programmablaufs nicht zu verwenden. Somit ist es notwendig, Programme, die mehrere Kerne verwenden sollen, aufzusplitten. Hier gilt es, Programmteile zu identifizieren, die unabhängig voneinander ausgeführt werden können ohne dass dabei das Ergebnis bzw. die Genauigkeit des Ergebnisses verfälscht wird. Diese Programmteile werden auch als Tasks bezeichnet. Hierbei kann man weiterhin zwischen verteilten Programmen und parallelen Programmen unterscheiden. Der Unterschied zwischen den beiden besteht darin, dass letztere auch auf nur einem Prozessor ausgeführt werden können, erstere jedoch nicht (Bengel et al., 2008, S. 24).

2.4.1.1 Allgemeine Charakteristika paralleler Algorithmen

Zur Charakterisierung paralleler Algorithmen können diverse Charakteristika herangezogen werden. Hier werden die Art der Dekomposition, die Granularität, der Parallelisierungsgrad sowie die Art der Kommunikation behandelt.

Die Dekomposition gibt an, inwiefern ein Algorithmus aufgesplittet wird. Man unterscheidet dabei zwischen einer funktionalen Zerlegung, einer Datenzerlegung und einer Kombination von beiden (Bengel et al., 2008, S. 324). Die funktionale Zerlegung gliedert einen Algorithmus nach Funktionen oder Arbeitsschritten. Diese können unabhängig von einander ausgeführt werden, „was dem inhärenten Parallelismus nahe kommt“[33] (Bengel et al., 2008, S. 324). Als Beispiel kann der GA genannt werden. Hierbei könnte die neue Generation durch unterschiedliche Crossover (Funktionen) gebildet werden. Diese Crossover hängen nicht voneinander ab, da nur mit bereits bekannten Daten, d.h. den Individuen der alten Population gearbeitet wird. Ein Iterationsschritt könnte so aussehen, dass jedem Crossover dieselben Eltern aus der Population zugeteilt und diese rekombiniert werden (vgl. Abbildung 2.19(a)). Der so erzeugte Nachwuchs geht dann in die Menge der erzeugten Individuen ein und wird eventuell für die neue Population gewählt. Eine Datendekomposition setzt hingegen an den Daten als solches an. Hierbei werden die Daten zerlegt und dieselbe Funktion parallel auf verschiedene Daten angewendet (Bengel et al., 2008, S. 325f.). Auch dies soll am Beispiel des GA verdeutlicht werden. Angenommen es steht ein Crossover (Funktion) zur Verfügung, und es sollen aus einer bestehenden Population Nachkommen gebildet werden. Eine Möglichkeit bestünde darin, dass der Crossover auf mehreren Prozessoren gleichzeitig ausgeführt wird. Dazu sucht sich jeder Crossover die Elternpaare unabhängig voneinander aus bzw. es werden jedem Eltern zugewiesen. Hier wird im Vergleich zum Beispiel zur Funktionsdekomposition, mit unterschiedlichen Daten (nämlich den Individuen bzw. Chromosomen) gearbeitet. Die Eltern werden zur Rekombination verwendet, d.h. auf den unterschiedlichen Daten wird jeweils die gleiche Funktion (Crossover 1) angewendet (vgl. Abbildung 2.19(b)). In Analogie dazu verhalten sich Kombinationen aus Funktions- und Datenzerlegung.

[33] Nach Bengel et al. (2008, S. 324) besitzt ein Problem einen inhärenten Parallelismus, wenn es „aus vielen Teilproblemen, die voneinander total unabhängig sind, d.h. zwischen den Teilproblemen besteht keine funktionale Abhängigkeit und die Datenbereiche der Teilprobleme sind nicht gemeinsam und getrennt“, besteht.



(a) Funktional (b) Datenbasiert

Abbildung 2.19: Dekomposition am Beispiel eines GA

Zwei Charakteristika, die bei Einordnung paralleler Algorithmen ebenfalls Verwendung finden und zusammenspielen, sind die Granularität und der Grad der Parallelisierung. Nach Rauber und Rünger (2012, S. 5) wird mit der Granularität die „durchschnittliche Größe der Teilaufgaben“, die durch die Dekomposition entstehen, gemessen. Als Maßstab hierzu kann bspw. die Anzahl der Instruktionen, die in einer Teilaufgabe ausgeführt werden, herangezogen werden. Den Grad der Parallelisierung kann man als Anzahl der Teilaufgaben, die parallel abgearbeitet werden, definieren (Rauber und Rünger, 2012, S. 6). In der Regel steigt die Anzahl der Teilaufgaben, wenn eine feinere Granularität gewählt wird, d.h. wenn nur wenige Instruktionen pro Task ausgeführt werden. Das führt jedoch oftmals dazu, dass ein höherer Verwaltungsaufwand nötig ist, um die Teilergebnisse wieder zu einem Gesamtergebnis zusammenzuführen. Somit ist einerseits ein hoher Grad an Parallelität gewünscht und dem entgegengesetzt ist der Wunsch nach einer möglichst groben Granularität (Rauber und Rünger, 2012, S. 6). Beim Design eines parallelen Algorithmus muss man sich also die Frage stellen, inwiefern man sinnvoll Teile des Algorithmus in kleinere Probleme zerlegen kann unter Beachtung der Größe, die die Teilprobleme später haben werden.

Ein weiteres wichtiges Differenzierungskriterium ist das der Kommunikation zwischen den Prozessen bzw. Teilaufgaben. Hier lässt sich zwischen synchroner und asynchroner Kommunikation unterscheiden. Liegt in dem parallelen Algorithmus eine synchrone Kommunikation vor, so müssen Teilaufgaben an bestimmten Stellen auf andere Teilaufgaben, die noch nicht beendet sind, aber zur weiteren Berechnung benötigt werden, warten. Demgegenüber steht die asynchrone Kommunikation, bei der die Teilprozesse durchaus miteinander Informationen austauschen, sich dabei aber nicht untereinander blockieren (Crainic und Toulouse, 2010, S. 504).

2.4.1.2 Charakteristika paralleler Metaheuristiken

Die bisher betrachteten Merkmale sind allgemeiner Natur und gelten für die meisten parallelen Algorithmen. Da sich die Arbeit jedoch mit Metaheuristiken für Tourenplanungsprobleme und deren Parallelisierung befasst, sollen nun spezifische Merkmale, die vorwiegend zur Beurteilung paralleler

Metaheuristiken dienen, betrachtet werden.[34]

In Analogie zu der Arbeit von Crainic und Toulouse (2010) sollen im Folgenden drei Dimensionen zur Einordnung paralleler Metaheuristiken vorgestellt werden. Dabei handelt es sich um:

1. Search Control Cardinality (SCC)
2. Search Control and Communications (SCaC)
3. Search Differentiation (SD)

Die SCC bestimmt, wie Informationen zwischen den verschiedenen Prozessen ausgetauscht werden bzw. wie der Austausch kontrolliert wird. Es stellt sich hierbei die Frage, ob die globale Suche, d.h. die Metaheuristik, mittels eines Prozesses, 1-control (1C), oder mittels mehrerer Prozesse, p-control (pC), die zusammenarbeiten können, gesteuert wird (Crainic und Toulouse, 2010, S. 504).

Die Dimension der SCaC zielt auf die Kommunikationseigenschaften paralleler Algorithmen, wie sie im vorangegangenen Abschnitt behandelt wurden, ab. Dabei stimmen die Autoren mit der allgemeinen Trennung zwischen synchroner und asynchroner Kommunikation überein. Jedoch sind sie der Ansicht, dass diese Einteilung nicht trennscharf genug ist, weshalb sie eine differenziertere Untergliederung in vier Teilbereiche vornehmen. Sie unterscheiden zwischen Rigid Synchronization (RS), Knowledge Synchronization (KS), Collegial (CO)[35], und Knowledge Collegial (KC). Unter RS verstehen Crainic und Toulouse (2010) ein Ablaufschema in dem die Prozesse zu bestimmten Zeiten ihre Ergebnisse den anderen Prozessen mitteilen. Als Ergebnisse sind im Kontext der Tourenplanung Lösungen von Probleminstanzen zu verstehen. Die KS geht in Analogie dazu vor, mit dem Unterschied, dass neben Lösungen weitere Informationen übertragen werden. Denkbar sind laut den Autoren bspw. statistische Auswertungen der Lösungen, um darauf aufbauend den Suchprozess im Folgenden weiter zu steuern. Die Klassen CO und KC sind die Pendants zu RS und KS mit dem Unterschied, dass die Prozesse nicht zu bestimmten Zeiten bzw. an bestimmten Punkten auf andere Prozesse warten müssen, um Informationen mitzuteilen oder zu erhalten.

Auf der Ebene der SD unterscheiden die Autoren zwischen „same initial point/population, same search strategies“ (SPSS), „same initial point/population, different search strategies“ (SPDS), „multiple initial points/populations, same search strategies“ (MPSS) und „multiple initial points/population, different search strategies“ (MPDS). Im Falle des SPSS startet der parallele Algorithmus mit einer Lösung. Auf diese Lösung wird jeweils die gleiche Suchstrategie angewendet, d.h. jeder der Prozesse evaluiert bspw. einen Teil der Lösung. Im Unterschied dazu steht SPDS, bei dem zwar vom gleichen Startpunkt ausgegangen wird, allerdings werden unterschiedliche Strategien von den Suchprozessen verwendet. So könnten in einer Metaheuristik wie der VNS von einer Initiallösung ausgehend mehrere Nachbarschaften parallel nach dem bestmöglichen Move durchsucht werden. Bei MPSS wird hingegen von mehreren Lösungen ausgegangen und jeweils die gleiche Suchstrategie angewendet. Als Beispiel werden hier verschiedene Lösungen generiert und von jeder Ausgangslösung bspw. das lokale Optimum mittels der gleichen Suchstrategie gesucht. Schließlich nennen die Autoren noch die MPDS, wobei hier von mehreren Lösungen ausgegangen wird, die jedoch unterschiedlich behandelt werden.

[34] Spezifisch bedeutet in diesem Fall nicht, dass sich die Merkmale nur auf Tourenplanungsprobleme bzw. parallele Metaheuristiken anwenden lassen. Dennoch werden die Merkmale von dieser Sichtweise her betrachtet.

[35] Die Autoren kürzen Collegial mit C ab. Da dieser Buchstabe bereits mit der Maximalkapazität in dieser Arbeit belegt ist, wird hier die Abkürzung CO verwendet.

Die Taxonomie von Crainic und Toulouse (2010) ermöglicht eine fein abgestimmte Differenzierung paralleler Algorithmen, weshalb sie zur Einordnung von den parallelen Verfahren, die in dieser Arbeit verwendet werden, genutzt werden soll.

2.4.2 Vergleich paralleler und sequentieller Algorithmen

Der vorherige Unterabschnitt beschäftigt sich mit Eigenschaften paralleler Algorithmen und der Möglichkeit ihrer Einordnung. Diese Charakteristika dienen vornehmlich dem Vergleich paralleler Verfahren untereinander. Ein paralleler Algorithmus entspringt jedoch oftmals aus einem zuvor sequentiellen Verfahren. Daher ist es nicht nur notwendig, parallele Algorithmen untereinander einordnen zu können, sondern auch den parallelen Algorithmus, der aus dem sequentiellen Verfahren entsteht, mit eben diesem vergleichen zu können. Dabei zielt der Vergleich zwischen parallel und sequentiell weniger auf strukturelle Eigenschaften, sondern viel mehr auf die Performance (Zeit) ab. Das ergibt sich direkt vor dem Hintergrund, dass das Ziel der Parallelisierung eine schnellere Ausführung eines Algorithmus ist. Die Beschreibung der folgenden Performancemaße orientiert sich an der Gliederung von Bengel et al. (2008, S. 313ff.). Dazu werden zunächst die Bestandteile, aus denen sich die Laufzeit eines Algorithmus zusammensetzt, beschrieben. Darauf aufbauend werden die Kennzahlen Speedup, Kosten, Overhead und Effizienz näher betrachtet, bevor anschließend auf Amdahls und Gustafsons Gesetz eingegangen wird. Der Abschnitt schließt mit einer kurzen Betrachtung der Karp-Flatt-Metrik.

2.4.2.1 Laufzeit paralleler Algorithmen

Als Basis vieler Performancemaße dient die Laufzeit $T_p(n)$ eines Algorithmus. Dabei entspricht n der Problemgröße und p der Anzahl der Prozessoren, die an der Problemlösung beteiligt sind (Bengel et al., 2008, S. 313). Sie setzt sich zusammen aus der Rechenzeit T_{CPU}, der Kommunikationszeit T_{COM}, der Wartezeit T_{WAIT}, der Synchronisationszeit T_{SYN}, der Platzierungszeit T_{Place} und der Startzeit T_{Start} (Bengel et al., 2008, S. 313). Die Rechenzeit ist die Zeit, die für die eigentlichen Berechnungen des Algorithmus benötigt wird. Hierbei fließen nur die Speicherzugriffe mit ein, die auf den lokalen Speicher des jeweiligen Prozessors ausgerichtet sind. Die Kommunikationszeit erfasst die Dauer, um Daten zwischen den beteiligten Prozessoren auszutauschen. Wartezeit entsteht, wenn die Dekomposition bzw. die Lastverteilung nicht adäquat durchgeführt wird, d.h. wenn ein Prozessor länger braucht als ein anderer und der Prozessor, der schneller fertig ist, auf den langsameren warten muss. Die Synchronisationszeit ist die Zeit, die benötigt wird, um die Prozessoren untereinander zu koordinieren. Die Platzierungszeit ist die Dauer, die bei der Verteilung der Aufgaben auf die beteiligten Prozessoren anfällt. Die Startzeit erfasst schließlich die Dauer bis alle Prozesse auf allen Prozessoren gestartet sind (Bengel et al., 2008, S. 313f.).

2.4.2.2 Speedup paralleler Algorithmen

Der Speedup wird von der Laufzeit bzw. den Laufzeiten eines Algorithmus in der parallelen und sequentiellen Variante determiniert:

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \tag{2.34}$$

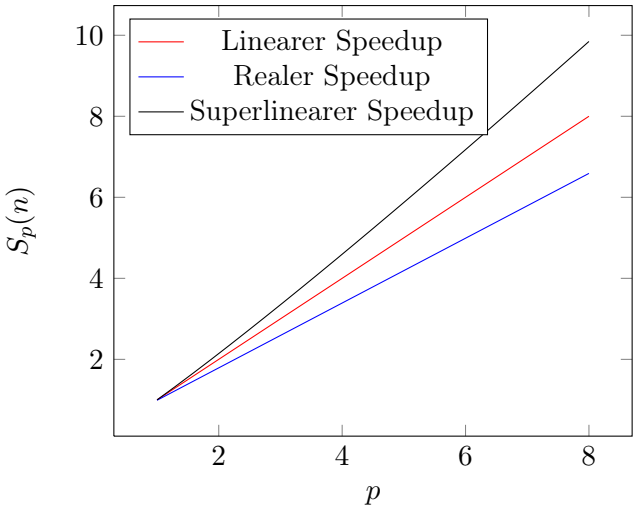


Abbildung 2.20: Speedup-Arten in Anlehnung an Bengel et al. (2008, S. 314)

$T^*(n)$ erfasst die Laufzeit der besten sequentiellen Implementierung des Algorithmus (Rauber und Rünger, 2012, S. 177). Der Speedup misst „den relativen Geschwindigkeitsvorteil [...], der gegenüber der besten sequentiellen Implementierung durch den Einsatz von Parallelverarbeitung auf p Prozessoren entsteht“ (Rauber und Rünger, 2012, S. 177). In der Theorie kann der Speedup maximal auf p ansteigen, da man in einem anderen Fall ein sequentielles Programm schreiben könnte, das die Schritte des parallelen Algorithmus nacheinander ausführt (Rauber und Rünger, 2012, S. 177). Dennoch kann der Fall eintreten, dass ein Speedup größer p erzielt wird. Dann spricht man von einem superlinearen Speedup. Das kann nach Rauber und Rünger (2012, S. 177f.) damit begründet werden, dass der beste sequentielle Algorithmus unbekannt ist oder die Implementierung aus praktischen Gründen (Aufwand) nicht umgesetzt wird. Des Weiteren lässt sich diese Art des Speedups oftmals damit erklären, dass die Cache-Strukturen durch die größere Anzahl von Prozessoren bzw. Threads besser genutzt werden (Rauber und Rünger, 2012, S. 178). In Abbildung 2.20 sind die drei unterschiedlichen Arten des Speedups visualisiert. In der Praxis hat man es in den meisten Fällen nicht mit dem Extremfall des superlinearen oder auch linearen Speedups zu tun, sondern mit dem realen Speedup. Der Geschwindigkeitsvorteil, der durch die Parallelisierung entsteht, wird in der Realität durch den Kommunikationsaufwand zwischen den Prozessoren oder die i.d.R. notwendige Synchronisierung abgeschwächt, sodass meist gilt $S_p(n) < p$.

2.4.2.3 Kosten und Overhead paralleler Algorithmen

Ein weiterer wichtiger Faktor zur Bewertung der Parallelisierung sind die Kosten, die sich nach folgender Formel ergeben:

$$C_p = T_p(n) \cdot p \tag{2.35}$$

Hierbei wird die Zeit berechnet, die der parallele Algorithmus in sequentieller Ausführung benötigt hätte (Bengel et al., 2008, S. 315). Das heißt, die Gesamtzeit eines parallelen Programms wird mittels $T_p(n)$ gemessen. Wenn nun p Prozessoren an der Problemlösung beteiligt sind, ergibt sich eine Gesamtzeit aus sequentieller Sicht, indem man die Parallelzeit mit der Anzahl der Prozessoren multipliziert. In diesem Zusammenhang lässt sich auch feststellen, ob ein paralleler Algorithmus kostenoptimal ist. Wenn ein

paralleles Programm so viele Operationen wie sein sequentielles Pendant ausführt, bezeichnet man es als kostenoptimal (Bengel et al., 2008, S. 315). Es gilt dann:

$$C_p(n) = T^*(n) \tag{2.36}$$

Hierauf aufbauend lässt sich der Overhead eines parallelen Verfahrens nach Bengel et al. (2008, S. 316) folgendermaßen bestimmen:

$$H_p(n) = C_p(n) - T^*(n) = p \cdot T_p(n) - T^*(n) \tag{2.37}$$

Es wird also die Differenz zwischen den Kosten des parallelen Programms, d.h. der theoretischen sequentiellen Laufzeit und der besten sequentiellen Implementierung gebildet. Wenn $H_p(n) > 0$ so bedeutet das, dass ein Overhead, der bspw. durch den Kommunikationsaufwand bei der Parallelisierung entsteht, vorliegt. Wenn $H_p(n) \leq 0$ ist, liegt kein Overhead vor (Ausnahme).

2.4.2.4 Effizienz paralleler Algorithmen

Ein Maß, das in die ähnliche Richtung deutet, ist die Effizienz eines parallelen Algorithmus, die wie folgt ermittelt wird (Bengel et al., 2008, S. 317):

$$E_p(n) = \frac{S_p(n)}{p} = \frac{T^*(n)}{p \cdot T_p(n)} = \frac{T^*(n)}{C_p(n)} \tag{2.38}$$

Im Unterschied zum Overhead wird hier Normierung vorgenommen. Wenn $E_p(n) < 1$, ist das Verfahren „suboptimal bezüglich seiner Kosten“[36], bei $E_p(n) = 1$ ist das Verfahren kostenoptimal und bei $E_p(n) > 1$ „liegt ein superlinearer Speedup vor“[37] (Bengel et al., 2008, S. 317).

2.4.2.5 Gesetze zur Prognose und Bewertung paralleler Algorithmen

Die bisher vorgestellten Überlegungen zielen darauf ab, im Nachhinein, wenn die Parallelisierung durchgeführt wurde, zu bestimmen, wie effizient die aufgewendete Arbeit ist. So kann es vorkommen, dass man erst nach der Parallelisierung feststellt, dass der zu parallelisierende sequentielle Algorithmus nicht oder nur in unzureichendem Maße von der Parallelisierung profitiert hat. Um bereits im Vorfeld der Implementierung abschätzen zu können, ob genügend Effizienzgewinne durch die Parallelisierung möglich sind, kann das Gesetz nach Amdahl (1967) herangezogen werden.[38] Angenommen, ein paralleles Verfahren besitzt einen sequentiellen Anteil f, der sich nicht parallelisieren lässt und der von der Problemgröße abhängt, mit $0 \leq f \leq 1$. Daraus resultiert der parallele Anteil mit $(1 - f)$ und es folgt die untere Schranke für die Laufzeit des parallelen Programms (Bengel et al., 2008, S. 317):

$$T_p(n) \geq f \cdot T^*(n) + \frac{(1-f) \cdot T^*(n)}{p} \tag{2.39}$$

[36] In diesem Fall ist $H_p(n) > 0$, d.h. es liegt ein Overhead vor.

[37] Für die letzten beiden Fälle ist $H_p(n) \leq 0$, d.h. es liegt kein Overhead vor.

[38] Hierbei geht es nur um eine obere Schranke (des Speedups), die durch die Parallelisierung erreicht werden kann. Das Gesetz prophezeit nicht die Fähigkeiten des Programmierers, der die Implementierung vornimmt. Es kann also in der Realität weiterhin zu starken Abweichungen kommen.

Nun lässt sich der maximale Speedup eines parallelisierten Programms ableiten:

$$\begin{aligned} S_p(n) &= \frac{T^*(n)}{T_p(n)} \\ &= \frac{T^*(n)}{f \cdot T^*(n) + \frac{(1-f)\cdot T^*(n)}{p}} = \frac{1}{f + \frac{1-f}{p}} \end{aligned} \tag{2.40}$$

Demnach ist die Laufzeit eines parallelen Algorithmus mindestens so groß wie die Laufzeit, die sich aus seinem sequentiellen Anteil ergibt, zuzüglich der Laufzeit, die aus dem parallelen Anteil resultiert. Der Speedup resultiert lediglich aus dem sequentiellen Anteil und der Anzahl der Prozessoren, d.h., die Problemgröße fließt in Amdahls Gesetz nicht ein. Wenn man nun annimmt, dass $p \to \infty$, dann ergibt sich ein maximal erreichbarer Speedup von $\frac{1}{f}$ (Rauber und Rünger, 2012, S. 179). So verlockend die Quantifizierung nach Amdahls Gesetz auch sein mag, so trügerisch kann sie sein. Hierbei sollte insbesondere beachtet werden, dass die Bestimmung des sequentiellen und parallelen Anteils oftmals nicht ohne erhebliche Probleme zu bewerkstelligen ist, und es somit zu Fehleinschätzungen bzgl. der Parallelisierbarkeit kommen kann. Weiterhin berücksichtigt das Gesetz nicht die Problemgröße (Bengel et al., 2008, S. 318), die in der Realität erheblichen Einfluss auf den resultierenden Speedup nimmt.

Um dem Umstand Rechnung zu tragen, dass nach Amdahls Gesetz die Problemgröße konstant gehalten wird, schlägt Gustafson (1988, S. 532) ein alternatives Maß vor, bei dem angenommen wird, dass die Problemgröße mit zunehmender Anzahl der Prozessoren ebenfalls ansteigt, d.h., der parallelisierbare Anteil wächst mit Problemgröße und Prozessorzahl. Daraus ergibt sich:

$$f = \frac{f_1}{p \cdot (1 - f_1) + f_1} \tag{2.41}$$

Dabei ist f_1 der „sequentielle Anteil des Problems bei einer festen Problemgröße, die in einer festen Zeit auf einem Prozessor abgearbeitet werden kann“ (Bengel et al., 2008, S. 319). Hieraus folgt unter Nutzung von Amdahls Gesetz das Gesetz von Gustafson für den Speedup:

$$S_p(n) = p \cdot (1 - f_1) + f_1 \tag{2.42}$$

Das Gesetz von Gustafson drückt also aus, „dass der Speedup mit einem konstanten sequentiellen Teil annähernd linear mit der Prozessorzahl wächst. Damit ist bei massiv parallelen Maschinen oder Clustern und großem p und entsprechend großen Problemgrößen ein Speedup nahe p möglich“ (Bengel et al., 2008, S. 319f.). Darauf aufbauend versteht man unter der Skalierbarkeit einer Anwendung, „dass die Effizienz eines parallelen Programmes bei gleichzeitigem Ansteigen von Prozessoranzahl p und Problemgröße n konstant gehalten wird“ (Rauber und Rünger, 2012, S. 180).

Was die beiden zuletzt vorgestellten Gesetze nicht beachten, ist der Overhead, der durch die Parallelisierung entsteht. Deshalb veröffentlichen Karp und Flatt (1990, S. 540) ein alternatives Maß, das auf empirischen Daten beruht und zur Beurteilung der Parallelisierung im Nachhinein verwendet werden kann:

$$f = \frac{\frac{1}{S_p(n)} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{2.43}$$

Die Metrik misst die Effizienz der Parallelisierung.[39] Dabei wird der Speedup gemessen und zur Berechnung der Kennzahl verwendet. Wenn f mit steigender Prozessorzahl p wächst, so kann das ein Zeichen dafür sein, dass die Granularität der Aufgaben zu klein gewählt ist. Weiterhin kann die

[39] Damit ist nicht die bereits vorgestellte Effizienz gemeint, sondern ein ergänzendes Effizienzmaß.

Metrik auf Hardwaregegebenheiten hinweisen, die den seriellen Anteil bzw. den Overheadanteil mit zunehmendem p ansteigen lassen.

Es gibt unzählige Arten zur Beurteilung des parallelisierten Programms (sowohl ex ante als auch ex post). Jede Maßzahl versucht etwas anderes zu messen, sodass sich in dieser Arbeit nicht nur auf eine der vorgestellten Maßzahlen beschränkt werden soll, sondern dass mehrere der vorgestellten Verfahren im gemeinsamen Zusammenhang betrachtet werden sollen (vgl. Kapitel 4.1).

2.5 Vergleich von Metaheuristiken

In Abschnitt 2.4 wurde auf die Möglichkeiten zur Charakterisierung von (parallelen) Algorithmen eingegangen. Die dabei genannten Merkmale zielen in erster Linie darauf ab, immer den gleichen Algorithmus zu untersuchen. Ebenso betrachtet der Vergleich zwischen paralleler und sequentieller Version im Grunde immer das gleiche Verfahren, das auf eine etwas andere Art implementiert wird.[40] Man kann bei dem Vergleich von Metaheuristiken zwischen qualitativen und quantitativen Merkmalen unterscheiden.

Unter qualitativen Merkmalen werden in dieser Arbeit die strukturellen Unterschiede im Aufbau des Algorithmus als solches verstanden. Als einfachstes Beispiel sei die Unterscheidung zwischen einer TS und einem GA genannt. Wie bereits erwähnt, ist diese Differenzierung jedoch sehr grob, sodass man in den meisten Fällen weiter differenzieren muss. Das heißt, es gibt bspw. im Bereich der Tourenplanung verschiedenste Metaheuristiken, die einen GA als Grundlage (vgl. Abschnitt 2.6) verwenden, jedoch haben diese nur wenig gemein. Anhand qualitativer Charakteristika lassen sich nur schwer Aussagen über die Güte eines Algorithmus anstellen. Um diesem Umstand zu begegnen, werden quantitative Merkmale herangezogen. Die Lösungsgüte und die Laufzeit eines Verfahrens stellen in dieser Arbeit die Grundlage für diesen Vergleich. Sie werden mit Hilfe von Benchmarkinstanzen ermittelt. Deshalb wird im Folgenden zunächst die Ermittlung der Lösungsgüte und der Laufzeit im Vordergrund stehen, bevor darauf folgend auf Benchmarks eingegangen wird.[41]

2.5.1 Lösungsgüte und Laufzeit zum Vergleich von Metaheuristiken

Beim Vergleich quantitativer Eigenschaften kommen bei Metaheuristiken grundsätzlich die Laufzeit und Lösungsgüte eines Algorithmus in Frage. Hier spiegelt sich auch ein wesentlicher Unterschied zu exakten Verfahren wider. Bei diesen spielt die Lösungsgüte nur eine untergeordnete Rolle, da sie immer die Optimallösung eines kombinatorischen Optimierungsproblems garantieren und man somit nur auf die Zeit als Beurteilungsmaßstab abstellen kann bzw. auf die maximal lösbare Instanzgröße. Barr et al. (1995, S. 14) schlagen sechs Kriterien vor, die zur Beurteilung eines Algorithmus herangezogen werden können:

1. die Qualität der besten gefundenen Lösung

2. die Laufzeit, bis die beste Lösung gefunden wurde

[40] Hier muss beachtet werden, dass man eine Metaheuristik auf viele verschiedene Arten umsetzen kann. Mit „gleiches Verfahren“ ist in diesem Zusammenhang ein exakt gleiches Verfahren mit dem Unterschied der Parallelisierung zu verstehen.

[41] Beide Themenblöcke werden jeweils vor dem Hintergrund des CVRPs betrachtet.

3. die Laufzeit, um gute Lösungen zu bestimmen
4. die Robustheit des Algorithmus
5. die Entfernung der besten Lösung zu guten Lösungen
6. der Tradeoff zwischen Gültigkeit und Lösungsgüte.

Alle Charakteristika beziehen sich auf die Lösungsgüte bzw. auf die Laufzeit. Deshalb soll im Folgenden zunächst auf die Eigenschaft der Lösungsgüte eingegangen werden, bevor danach die Laufzeit betrachtet wird. Die genannte Robustheit und der Tradeoff zwischen Gültigkeit und Lösungsgüte sind für diese Arbeit nicht von Bedeutung, weshalb sie nicht weiter berücksichtigt werden sollen.

Um die Lösungsgüte eines Algorithmus[42] einordnen zu können, herrscht die einhellige Meinung, dass dies im Idealfall über die Optimallösung der betrachteten Probleminstanz erfolgen sollte (vgl. bspw. Barr et al. (1995, S. 14f.) oder Silberholz und Golden (2010, S. 634)). Das Problem, das hieraus resultiert, ist, dass gerade für große Probleminstanzen keine Optimallösungen existieren. Man kann mit Silberholz und Golden (2010, S. 634) übereinstimmen, dass Optimallösungen i.d.R. nur für kleine Probleminstanzen bekannt sind und es eben das Ziel von Metaheuristiken ist, große Instanzen, für die exakte Verfahren nicht mehr in Frage kommen, zu lösen. Das heißt, es gilt Wege zu finden, wie man eine Einordnung der Lösungsgüte ohne Optimallösung vornehmen kann. Ein Weg, der in dieser Arbeit jedoch nicht weiter betrachtet wird, führt über untere bzw. obere Schranken (Barr et al., 1995, S. 15). Eine andere Möglichkeit ist der Vergleich der Lösungsgüte des zu begutachtenden Verfahrens mit der Lösungsgüte der bisher besten bekannten Lösung.[43] Hier wird entweder die Abweichung zur besten bekannten Lösung bestimmt oder die Abweichung zwischen zwei Algorithmen, die direkt miteinander verglichen werden sollen (Silberholz und Golden, 2010, S. 634). Im letztgenannten Falle ergibt sich die Problematik, dass sich der Vergleich nur auf einen oder mehrere Algorithmen bezieht und sich keine Aussage über die Lösungsgüte direkt treffen lässt. Das resultiert daher, dass durch eine „falsche“ Auswahl die zu beurteilenden Algorithmen falsch eingeordnet werden, d.h. es könnte der Fall eintreten, dass ein Verfahren im Vergleich das beste ist, es jedoch eine sehr große Abweichung zur Optimallösung hat (Silberholz und Golden, 2010, S. 635). In dieser Arbeit werden beide Varianten verwendet, wobei letztgenannte immer auch vor dem Hintergrund der besten bekannten Lösungen betrachtet werden.

Beim Einsatz von Metaheuristiken wird bewusst darauf verzichtet, dass optimale Lösungen garantiert werden. Dieser Verzicht geht einher mit dem Wunsch nach kürzeren Laufzeiten im Vergleich zu exakten Verfahren bzw. mit dem Wunsch nach Lösung großer Problemklassen, die durch exakte Verfahren mit nicht vertretbarem Aufwand zu bearbeiten sind. Deshalb spielt bei einem Vergleich von Metaheuristiken die Laufzeit eine herausragende Rolle. Um Laufzeiten zu vergleichen, ist die sauberste Lösung, die zu vergleichenden Algorithmen in der gleichen Programmiersprache zu schreiben, mit dem gleichen Compiler (und gleichen Compiler-Parametern) zu übersetzen und auf demselben Rechner mit den gleichen Probleminstanzen auszuführen (Silberholz und Golden, 2010, S. 635). Allerdings ist dieses in den meisten Fällen nicht durchführbar, da der Quellcode der zu vergleichenden Algorithmen nicht zur Verfügung steht oder bspw. in einer anderen Programmiersprache vorliegt. Ein möglicher Ansatz wäre,

[42] Wenn in diesem Abschnitt von Algorithmus, Verfahren, Methode etc. die Rede ist, so ist stets ein (meta-)heuristisches Verfahren gemeint. Exakte Verfahren bleiben hier ausgeklammert bzw. werden nur bei expliziter Benennung berücksichtigt.

[43] Natürlich kann die bisher beste bekannte Lösung einer Probleminstanz auch als Schranke angesehen werden.

den Vergleichsalgorithmus ebenfalls zu implementieren, allerdings ergeben sich daraus weitere Probleme wie z.B., dass bei der Beschreibung des Vergleichsverfahrens Details fehlen und dass der so implementierte Code sich erheblich vom Originalcode unterscheidet (Silberholz und Golden, 2010, S. 635). Eine andere Möglichkeit besteht darin, die Ergebnisse, die andere Autoren veröffentlicht haben, direkt zu verwenden. Die minimale Voraussetzung dafür ist, dass die Rechnerkonfiguration auf der getestet wird, auch erwähnt wird. Das bringt einerseits den Vorteil, dass man sich nicht um die Implementierung kümmern muss und andererseits den Nachteil, dass die Vergleichsergebnisse längst nicht so exakt sind (Silberholz und Golden, 2010, S. 637). Trotz der Nachteile kommt in dieser Arbeit letztgenanntes Verfahren zum Einsatz, da der Aufwand der Reimplementierung in keinem Nutzenverhältnis steht. Um den Vergleich durchzuführen, benötigt man ein Mapping der CPUs, d.h., ein Vergleich von Laufzeiten von zwei Algorithmen zwischen denen mehrere Jahre liegen, kann nicht ohne einen adäquaten Umrechnungsfaktor vorgenommen werden. Dazu wird oftmals der Linpack-Benchmark von Dongarra (2011) verwendet. Hierbei werden für die unterschiedlichsten Rechner die erzielten MFlop/s[44] angegeben, die die Grundlage zur Umrechnung der Laufzeiten bilden. In Abschnitt 2.5.2 kommt dieses Vorgehen zum Einsatz.

Silberholz und Golden (2010, S. 637f.) schlagen noch zwei alternative Maße zur Laufzeitbeurteilung vor, nämlich die Wachstumsrate der Laufzeit bei steigender Problemgröße und die Anzahl der Operationen, die ein Algorithmus benötigt. Beide sind mehr oder minder unabhängig von der Hardware, es gibt jedoch bei letzterem das Problem, dass wichtige Operationen identifiziert werden müssen, was nicht ohne weiteres möglich ist. So stellt sich dabei die Frage, was ist eine wichtige Operation? Die Wachstumsrate der Laufzeit ist hingegen robuster, da sie unabhängig von der Programmiersprache und auch unabhängig von der verwendeten Hardware ist. Sie zeigt eine Tendenz wie sich ein Algorithmus bei unterschiedlicher Problemgröße verhält. Das Maß kommt in dieser Arbeit ebenfalls zur Anwendung. Allerdings lässt sich aus vielen Arbeiten die Wachstumsrate nicht ermitteln, weshalb hier nur das Verhalten der implementierten Algorithmen aufgezeigt wird und die Ergebnisse so von anderen Autoren in Zukunft verwendet werden können.

2.5.2 Benchmarkinstanzen des CVRPs

Der vorangegangene Abschnitt behandelt die Lösungsgüte und die Laufzeit von Algorithmen im Allgemeinen. Um diese Kennzahlen zu erhalten, ist es nötig, Testinstanzen zu verwenden. Deshalb sollen an dieser Stelle die verwendeten Testinstanzen und ihre Merkmale näher betrachtet werden.

Man kann grundsätzlich mit vorhandenen oder neuen Testinstanzen einen Algorithmus testen (Silberholz und Golden, 2010, S. 626). Die Generierung von neuen Problemdaten kommt insbesondere bei neuen Problemstellungen zum Einsatz, die bisher noch keine Berücksichtigung in der Literatur finden oder, um bspw. größere Probleme zu generieren. Letztgenanntes geschieht vor dem Hintergrund, dass durch die Evolution der Rechner kleinere Benchmarks auch mit exakten Verfahren in Sekundenbruchteilen berechnet werden können und somit Metaheuristiken für diese Instanzen obsolet sind. Da das CVRP ein

[44] Als alternatives Maß kommen noch die Million Instructions per Second (MIPS) in Frage. Die Maßzahl wäre dann für vorliegende Arbeit von Bedeutung, wenn bspw. die Anwendung von Moves den Hauptanteil an den Berechnungen hätten. Allerdings muss vor Anwendung eines Moves auf eine Lösung zunächst die vollständige Nachbarschaft abgesucht und alle Moves bewertet werden. Diese Bewertung erfolgt durch Fließkommazahlen, weshalb die MFlop/s für diese Arbeit das bessere Maß zum Vergleich darstellt. Gleiches gilt auch für einen Großteil der Verfahren der Literatur, da i.d.R. erst große Teile der Nachbarschaft bewertet werden, bevor letztendlich ein Move angewendet wird.

(a) Problem # 3 (b) Problem # 12

Abbildung 2.21: Visualisierung von Probleminstanzen von Christofides et al. (1979)

in der Literatur stark beachtetes Forschungsfeld ist, ist es nicht notwendig, neue Instanzen zu generieren, sondern es können bereits bestehende Probleme, auf die im Folgenden eingegangen wird, verwendet werden.

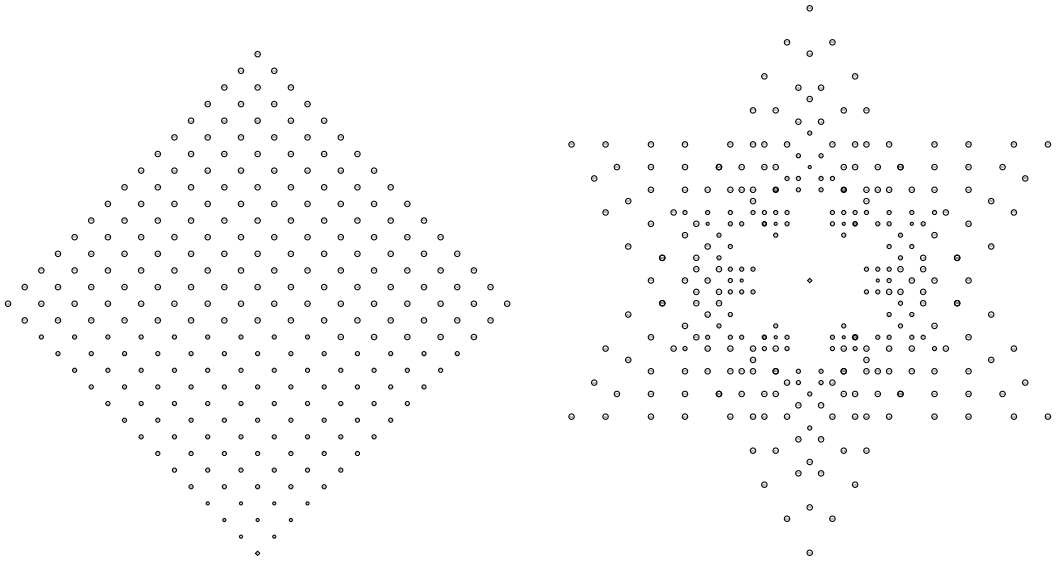
Es sollte noch angemerkt werden, dass alle verwendeten Instanzen in einem 2D-Raum aufbauen. Dazu werden in jeder Instanz die Koordinaten der Kunden und des Depots angegeben, woraus dann mittels der euklidischen Distanz die Abstände c_{v_i,v_j} zwischen den Kunden bzw. zwischen Kunden und Depot bestimmt werden. Weiterhin sichert die Nutzung der euklidischen Distanz zu, dass die Dreiecksungleichung erfüllt ist. Das heißt, die Bedingung $c_{v_i,v_z} + c_{v_z,v_j} \geq c_{v_i,v_j}, \forall v_i, v_j, v_z \in V$ schließt aus, dass Umwege bei der Routenberechnung berücksichtigt werden müssen.

Weiterhin sollte berücksichtigt werden, dass in vielen Publikationen, die sich mit dem CVRP beschäftigen, gleichzeitig Instanzen, die eine maximale Routendauer/distanz als Restriktion aufweisen (Distance-Constrained Capacitated-Vehicle-Routing-Problem), getestet werden. Die vorliegende Arbeit behandelt jedoch nur CVRP-Instanzen mit Kapazitätsbeschränkung, sodass im Folgenden der Fokus auf den Instanzen der jeweiligen Problemklasse liegt, die nur eine Kapazitätsbeschränkung aufweisen.

Die Instanzen von Christofides et al. (1979) bauen auf den Problemdaten von Christofides und Eilon (1969) auf und erweitern diese. Insgesamt enthält der Benchmark 14 unterschiedliche Instanzen. Die Größe der Probleminstanzen variiert von 50-199 Kunden. Weiterhin sind sieben der 14 Probleminstanzen distanzbeschränkt, d.h. diese sind nicht geeignet, um Verfahren für das CVRP zu testen. Somit werden in dieser Arbeit lediglich sieben Instanzen für numerische Analysen eingesetzt. Diese Instanzen lassen sich wiederum nach der Anordnung der Kunden charakterisieren. In den Problemen, die auf den Instanzen von Christofides und Eilon (1969) aufbauen, ist eine zufällige Anordnung der Kunden im 2D-Raum gegeben (vgl. Abbildung 2.21(a)). Christofides et al. (1979, S. 334) sehen darin jedoch die Praxis zu schlecht abgebildet, womit sie die Generierung zweier[45] neuer Probleminstanzen begründen, bei dem Kunden geclustert im 2D-Raum angeordnet sind (vgl. Abbildung 2.21(b)).

Die von Taillard (1993) vorgeschlagenen 15 Benchmarks enthalten 75-385 Kunden. Auch hier wird

[45] Es werden insgesamt vier neue Probleminstanzen gebildet, allerdings sind zwei davon distanzbeschränkt; von der Kundenanordnung hingegen sind sie identisch.



(a) Problem # 9 (b) Problem # 18

Abbildung 2.22: Visualisierung von Probleminstanzen von Golden et al. (1998)

im Unterschied zu den ersten fünf Problemen von Christofides et al. (1979) von einer ungleich verteilten Kundenstruktur ausgegangen, d.h., Kunden befinden sich geclustert im Raum (Alba und Dorronsoro, 2006, S. 227). Das besondere an der Instanz mit 385 Kunden ist, dass es sich um reale Daten bzw. Orte aus dem Schweizer Kanton Waadt handelt. Lediglich die Nachfrage der Orte wird künstlich generiert, weshalb diese Instanz auch als pseudo-real bezeichnet wird (Taillard, 1993, S. 671). Im Unterschied zu den zuvor vorgestellten Benchmarks enthalten die Probleme von Taillard (1993) keine Instanzen, bei denen die Distanz beschränkt ist, weshalb alle in dieser Arbeit zur Anwendung kommen.

Die wohl bekanntesten und am weitesten verbreiteten Benchmarks im Kontext des CVRP der vergangenen Jahre stammen von Golden et al. (1998). Die Kundenanzahl beläuft sich auf 200-483, weshalb die Instanzen oftmals auch als Large-Scale-Instanzen bezeichnet werden. Die Motivation zu größeren Instanzen ist dabei praxisorientiert, da die geringe Kundenanzahl nicht alle Praxisfälle abdeckt (Golden et al., 1997, S. 8). Insgesamt enthält die Problemklasse 20 Instanzen, wobei acht Instanzen (Nummer 1-8) mit Distanzbeschränkungen enthalten sind, die keine weitere Beachtung finden sollen. Ein wesentlicher Unterschied dieser Instanzen zu den anderen bisher vorgestellten ist die stark strukturierte Verteilung der Kunden im 2D-Raum. Hintergrund dessen ist, wie bereits in Abschnitt 2.5.1 erwähnt, dass große Probleme, zu denen man die Instanzen von Golden et al. (1998) zählen kann, nicht exakt gelöst werden können. Da die Instanzen stark strukturiert sind, lassen sich gute, wenn auch nicht gleichbedeutend mit den besten, Lösungen per Anschauen abschätzen (vgl. Golden et al. (1997, S. 18) oder Li et al. (2005, S. 1166)). In Abbildung 2.22 kann man die geometrischen Eigenschaften einiger Instanzen erkennen. Ein Nachteil der Probleme besteht darin, dass man wohl mit Sicherheit sagen kann, dass es sich nicht um eine Strukturierung von Kunden handelt, die in der Realität zu beobachten ist. Trotzdem haben sich die Golden-Instanzen[46] in den vergangenen Jahren in der Literatur durchgesetzt und werden in nahezu jedem Beitrag, der sich mit dem CVRP beschäftigt, herangezogen, um den Algorithmus daran zu messen. Deshalb bilden sie die Basis für einen Großteil der numerischen

[46] Auch wenn die Benchmarks zumeist von mehreren Autoren publiziert werden, werden die in dieser Arbeit verwendeten Daten auch als Golden-, Christofides-, Taillard- und Gehring-Instanzen bezeichnet.

Auswertungen, die in dieser Arbeit vorgestellt werden.

Dem Autor der vorliegenden Arbeit ist kein Datensatz für das CVRP bekannt, der eine größere Kundenzahl als die Golden-Instanzen enthält.[47] Aufgrund dessen werden zusätzliche Probleme, die im Kontext des VRPTW Anwendung finden, betrachtet. Diese stammen von Gehring und Homberger (1999) und bauen auf den Problemen von Solomon (1987) auf. Die Zeitfensterrestriktionen sollen in dieser Arbeit nicht berücksichtigt werden. Die Testdaten lassen sich in drei Klassen (R, C, RC) einteilen, die angeben, wie die Kunden verteilt sind. Im Falle der R-Klasse sind die Knoten zufällig auf der Fläche verteilt, wohingegen bei C-Instanzen die Kunden geclustert sind. Die RC-Instanzen enthalten beide Merkmale. Abbildung 2.23 visualisiert beispielhaft die unterschiedlichen Problemklassen. Es stellt sich hier berechtigterweise die Frage, warum nicht auf den Instanzen von Li et al. (2005) aufgebaut wird und dort die Distanzrestriktion relaxiert wird. Begründet wird das dadurch, dass dem Autor keine Verfahren bekannt sind, die eben dieses durchführen, wodurch die Lösungsgüte und Laufzeit nur eine begrenzte Aussagekraft hätten. Anders liegt der Fall bei den Instanzen von Gehring und Homberger (1999) bei denen Mester und Bräysy (2007) die Zeitfensterbedingung ebenfalls relaxieren und sich somit aussagekräftigere Schlussfolgerungen ableiten lassen.

Es existieren noch viele weitere Benchmarks, die sich mit dem CVRP beschäftigen, wie z.B. von Christofides und Eilon (1969) oder van Breedam (2002). Allerdings haben sich diese Benchmarks nicht durchgesetzt, weshalb auch in dieser Arbeit nicht näher darauf eingegangen wird.

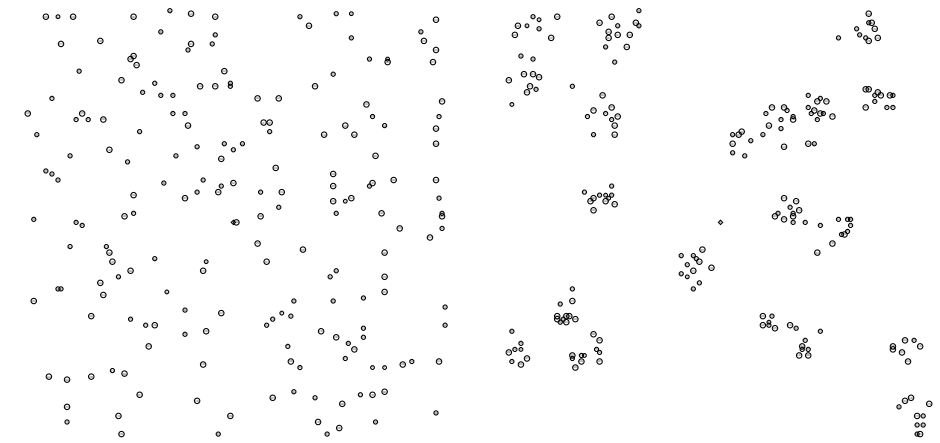
Nachdem die Benchmarks, die in diesem Beitrag verwendet werden, skizziert sind, soll nun auf die Lösungsgüten, die Verfahren der Literatur im Kontext der Testdaten erzielen, eingegangen werden. Die Benchmarkinstanzen von Golden et al. (1998) stehen im Fokus der Betrachtungen. Von den insgesamt 20 vorgeschlagenen Instanzen werden in dieser Arbeit die Probleme g9-g20 analysiert, da es sich bei g1-g8 um Instanzen mit zusätzlicher Distanzrestriktion handelt.

Es wird versucht, eine möglichst breite Palette an Heuristiken/Metaheuristiken der Literatur zur Lösung des CVRPs im Rahmen der Golden-Instanzen zu berücksichtigen. In Tabelle 2.1 sind alle in der Arbeit betrachteten Verfahren angegeben.[48] [49] Die Tabelle enthält weiterhin die MFlop/s, die der jeweilig eingesetzte Prozessor in etwa besitzt. Dazu werden die Daten von Dongarra (2011)

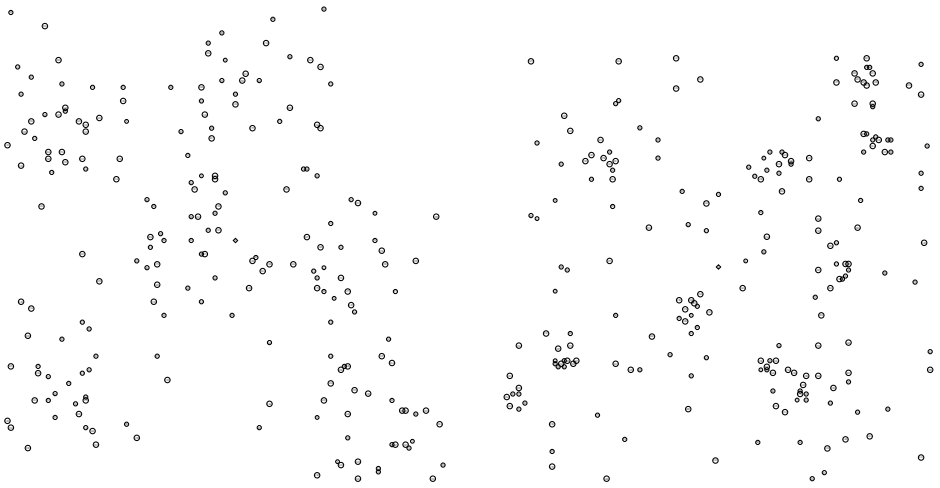
[47] Viele Arbeiten, die sich mit dem CVRP beschäftigen, behandeln gleichzeitig auch die distanzbeschränkte Variante davon. Das erklärt, warum es eine Vielzahl von Problemklassen gibt, die sowohl distanz- und kapazitätsbeschränkte Instanzen als auch nur kapazitätsbeschränkte Varianten enthalten. Dadurch hat sich in den vergangenen Jahren ein weiterer Benchmark-Satz zum Quasi-Standard-Satz herauskristallisiert. Er stammt von Li et al. (2005) und baut ebenfalls auf einer geometrischen Kundenverteilung auf. Die Motivation dieser Benchmarks liegt darin begründet, dass die Instanzen eine noch größere Anzahl von Kunden (bis zu 1200) enthalten. Ein wesentlicher Nachteil ist jedoch, dass die Instanzen distanz- und kapazitätsbeschränkt sind, d.h. Probleme, die sich nur auf die Kapazität beschränken, existieren nicht. Deshalb sollen diese Instanzen nicht weiter berücksichtigt werden.

[48] Aufgrund der Fülle an Artikeln zum VRP kann nicht garantiert werden, dass sämtliche Verfahren, die das CVRP im Zusammenhang mit den Instanzen von Golden et al. (1998) betrachten, enthalten sind. Allerdings sollten die wichtigsten Vertreter im Sinne der besten Lösungsqualität, die bis heute erzielt wird, enthalten sein. Weiterhin muss darauf hingewiesen werden, dass die Arbeit von Pisinger und Ropke (2007) nicht enthalten ist, da aus ihr nicht die Ergebnisse der einzelnen Instanzen extrahiert werden können. Die Heuristik als solche erzielt jedoch auf unterschiedlichsten Problemklassen sehr gute Resultate.

[49] Es muss noch angemerkt werden, dass es sehr viel mehr Publikationen im Bereich des CVRPs gibt. Allerdings werden diese hier nicht berücksichtigt, da in ihnen keine Ergebnisse für die Instanzen von Golden et al. (1998) veröffentlicht werden.



(a) Problem einer R-Instanz (200 Kunden) (b) Problem einer C1-Instanz (200 Kunden)



(c) Problem einer C2-Instanz (200 Kunden) (d) Problem einer RC-Instanz (200 Kunden)

Abbildung 2.23: Visualisierung von Probleminstanzen von Gehring und Homberger (1999)

herangezogen. Mittels der MFlop/s kann dann der Zeitfaktor[50] ermittelt werden. Dieser Zeitfaktor wird dazu verwendet, um die Zeiten, die in den Publikationen zum Lösen der Probleminstanzen angegeben sind, zu multiplizieren und somit eine grobe Vergleichbarkeit hinsichtlich der Rechenzeiten zwischen den Heuristiken herzustellen.[51] Als Referenz dient ein Intel Xeon 3,6 GHz. Alle anderen Rechner werden also auf diese Basis umgerechnet.

Ein Großteil der aufgelisteten Verfahren verwendet zur Implementierung ihres Algorithmus die Programmiersprache C/C++, was die Bedeutung der Sprache gerade im Bereich der Entwicklung von hochperformanten Algorithmen unterstreicht. Weiterhin kann man erkennen, dass die meisten Zeitfaktoren kleiner eins sind, was bedeutet, dass die Autoren mit einem entsprechend langsameren Rechner als dem Referenzrechner gearbeitet haben. Lediglich einige wenige Verfahren haben schnellere

[50] Der Zeitfaktor und die MFlop/s sind in der Tabelle auf zwei Stellen gerundet. Die weiteren Berechnungen mit diesen Kennzahlen basieren jedoch auf den exakten Werten.

[51] Es sollte hierbei beachtet werden, dass es sich nur um Annäherungen handelt, die keinesfalls der Realität entsprechen müssen. Insbesondere dann, wenn ein älteres Verfahren bzw. älterer Prozessor auf heutige Hardware projiziert wird, sind die Annäherungen mit großer Vorsicht zu verwenden.

Autoren	Programmiersprache	MFlop/s	Zeitfaktor	Abweichung [%]
Alabas-Uslu und Dengiz (2011)	Pascal	468,24	0,26	1,59
Alba und Dorronsoro (2006)	Java	1317,00	0,74	2,32
Chen et al. (2010)	C++	1378,15	0,77	1,16
Cordeau und Maischberger (2012)	C++	1458,59	0,82	0,80
Ergun et al. (2006)	-	229,17	0,13	1,46
Groër et al. (2010)	C++	1447,95	0,81	1,56
Groër et al. (2011)	C++	69191,67	38,89	0,02
Ho und Gendreau (2006)	C++	1190,00	0,67	2,10
Jin et al. (2011)	C++	2338,58	1,31	0,14
Jin et al. (2012)	C++	2986,88	1,68	0,51
Kwon et al. (2007)	C	1593,00	0,90	4,47
Kytöjoki et al. (2007)	C++	1386,67	0,78	7,37
Lee et al. (2010)	-	1571,00	0,88	2,55
Li et al. (2005)	-	465,00	0,26	1,53
Lin et al. (2009)	C	1317,00	0,74	1,65
Marinakis (2012)	Fortran 90	409,20	0,23	1,89
Marinakis und Marinaki (2010)	Fortran 90	409,20	0,23	1,00
Mester und Bräysy (2007)	Visual Basic	1317,00	0,74	0,35
Nagata (2007)	C++	1593,00	0,90	4,41
Nagata und Bräysy (2008)	C++	1593,00	0,90	0,14
Nagata und Bräysy (2009)	C++	1300,00	0,73	0,18
Prins (2004)	Delphi 5	250,80	0,14	2,23
Prins (2009)	Delphi 7	1317,00	0,74	0,82
Reimann et al. (2004)	C	185,21	0,10	1,70
Szeto et al. (2011)	C++	803,66	0,45	2,04
Tarantilis (2005)	C++	82,88	0,05	1,09
Toth und Vigo (2003)	Fortran 77	38,00	0,02	2,52
Turky (2011)	C++	986,05	0,55	1,47
Vidal et al. (2011)	C++	1571,00	0,88	0,00
Referenz		1779	1,0	

Tabelle 2.1: Verfahren zur Lösung der Golden-Instanzen

Hardware verwendet. Beim Spitzenreiter Groër et al. (2011) resultiert das hohe Ergebnis aus der Verwendung eines parallelen Algorithmus, der auf einem Rechencluster ausgeführt wird.

Aus diesen Daten lassen sich die angepassten Zeiten, die für jede Instanz von den Verfahren benötigt wird, extrahieren. In Tabelle B.3 im Anhang finden sich alle Verfahren mit ihrer jeweils besten gefundenen Lösung für die Golden-Instanzen g9-g20 und der angepassten Laufzeit. Hier ist anzumerken, dass sich auf die beste Lösung beschränkt wird. Das ist damit zu erklären, dass viele der Autoren lediglich die beste Lösung, jedoch keine Durchschnittslösung angegeben haben, wodurch bei einem Vergleich der Lösungsgüte auf Basis durchschnittlicher Werte keine aussagekräftigen Resultate hätten erzielt werden können. Mittels dieser Tabelle werden die besten bekannten Lösungen für die Instanzen von Golden et al. (1998) ermittelt und in Tabelle 2.2 dargestellt.

Instanz	Knotenanzahl	Lösungsgüte
g9	256	579,71
g10	324	736,26
g11	400	912,84
g12	484	1102,69
g13	253	857,19
g14	321	1080,55
g15	397	1337,92
g16	481	1612,50
g17	241	707,76
g18	301	995,13
g19	361	1365,60
g20	421	1818,25
Summe		13106,40

Tabelle 2.2: Beste bekannte Lösungen der Golden-Instanzen

Aus den besten bekannten Lösungen ergibt sich die Abweichung eines Verfahrens in Prozent, die ebenfalls in Tabelle 2.1 abgedruckt ist. Hier zeigt sich, dass der GA von Vidal et al. (2011) die besten Resultate für das CVRP erzielt. Dabei ist noch anzumerken, dass das Verfahren von den Autoren

eigentlich für Multi-Depot-Vehicle-Routing-Probleme (MDVRP) und Periodic-Vehicle-Routing-Probleme (PVRP) vorgeschlagen wird und das CVRP nur am Rande in der Arbeit getestet wird. Außerdem werden sehr gute Ergebnisse von Groër et al. (2011) mit der Kombination eines Randomized Record-to-Record-Travel-Algorithmus (RRTR) und einem exakten Verfahren, die auf einem Rechencluster ausgeführt werden, erzielt. Auch Jin et al. (2011, 2012) erreichen sehr gute Resultate und verwenden hierbei eine parallele TS, ebenso wie Cordeau und Maischberger (2012). Mester und Bräysy (2007), Nagata und Bräysy (2008, 2009) sowie Prins (2009) bauen in ihren sehr erfolgreichen Verfahren jeweils auf einem evolutionären Algorithmus auf.[52] Grundsätzlich lässt sich hier die Tendenz erkennen, dass Algorithmen, die parallelisiert sind bzw. Algorithmen, die einen inhärenten Parallelismus besitzen (Evolutionäre Algorithmen), die besten Resultate auf den Benchmarks von Golden et al. (1998) erzielen. Abbildung 2.24 zeigt die Positionierung der einzelnen Verfahren in Abhängigkeit der kumulierten Zeit (in Sekunden) und Kosten (Zielfunktionswert) über die Instanzen g9-g20.[53] Ziel bei dem Entwurf einer neuen Metaheuristik muss sein, dass sie sich möglichst weit unten links einordnet.

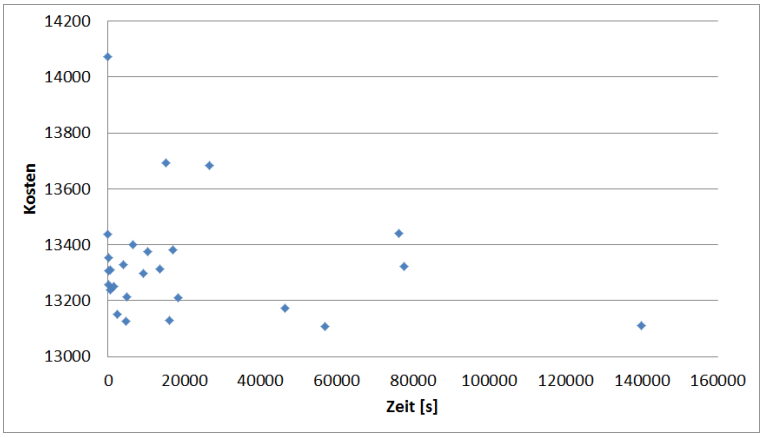


Abbildung 2.24: Positionierung der Heuristiken

Ferner werden in dieser Arbeit die Probleminstanzen c1-c5 und c11-c12 von Christofides et al. (1979) betrachtet. Die besten bekannten Lösungen dieser Instanzen in Tabelle 2.3 sind von Groër et al. (2011, S. 325) übernommen. Die anderen 7 Instanzen der Problemklasse enthalten wieder eine Distanzrestriktion und werden deshalb nicht berücksichtigt.

Instanz	Knotenanzahl	Lösungsgüte
c1	51	524,61
c2	76	835,26
c3	101	826,14
c4	151	1028,42
c5	200	1291,29
c11	121	1042,11
c12	101	819,56
Summe		6367,39

Tabelle 2.3: Beste bekannte Lösungen der Christofides-Instanzen

Im Vergleich zu den bisherigen Instanzen sind die Probleme von Taillard (1993) lediglich kapazitätsbeschränkt. In Tabelle 2.4 sind die besten bekannten Lösungen aufgelistet (entnommen aus Nagata und

[52] Viele der genannten Verfahren, werden im folgenden Abschnitt näher erläutert.

[53] In der Abbildung fehlen die Ergebnisse von Alba und Dorronsoro (2006) und Turky (2011), da sie keine Zeitangaben veröffentlichen sowie das Verfahren von Jin et al. (2012), da dieses eine sehr lange Zeit benötigt, die Grafik dadurch sehr stark verzerrt wird und die Positionierung der meisten Verfahren nicht mehr klar ersichtlich ist.

Bräysy (2009, S. 211) und Groër et al. (2011, S. 325)).

Instanz	Knotenanzahl	Lösungsgüte
T75a	76	1618,36
T75b	76	1344,62
T75c	76	1291,01
T75d	76	1365,42
T100a	101	2041,34
T100b	101	1939,90
T100c	101	1406,20
T100d	101	1580,46
T150a	151	3055,23
T150b	151	2727,20
T150c	151	2341,84
T150d	151	2645,40
T385	386	24366,69
Summe		47723,67

Tabelle 2.4: Beste bekannte Lösungen der Taillard-Instanzen

Letztlich fehlen noch die Instanzen von Gehring und Homberger (1999), die für das VRPTW gedacht sind, bei denen jedoch in dieser Arbeit auf die Zeitfensterrestriktion verzichtet wird. Bisher ist lediglich Mester und Bräysy (2007) in ähnlicher Weise vorgegangen, weshalb die besten bekannten Lösungen (vgl. Tabelle 2.5) von diesen Autoren stammen. Die Tatsache, dass die Instanzen erst mittels eines Verfahrens betrachtet werden, schmälert einerseits die Aussagekraft bei Vergleichen mit diesen Problemen. Andererseits hat sich der Algorithmus von Mester und Bräysy (2007) als sehr gut bei Betrachtung anderer Benchmarkklassen erwiesen, weshalb die Resultate, die hier erzielt werden, nicht zu weit von den besten Lösungen entfernt sein sollten.

Instanz	Knotenanzahl	Lösungsgüte	Instanz	Knotenanzahl	Lösungsgüte
C1_200	201	2570,49	R2_200	201	1610,30
C1_400	401	6765,03	R2_400	401	3323,05
C1_600	601	13465,24	R2_600	601	6254,17
C1_800	801	23999,35	R2_800	801	10230,06
C1_1000	1001	39811,15	R2_1000	1001	15149,40
C2_200	201	1368,89	RC1_200	201	2804,20
C2_400	401	2799,40	RC1_400	401	7381,47
C2_600	601	5441,98	RC1_600	601	14786,15
C2_800	801	8241,83	RC1_800	801	26720,23
C2_1000	1001	11792,66	RC1_1000	1001	41583,54
R1_200	201	2878,24	RC2_200	201	1513,00
R1_400	401	7192,03	RC2_400	401	3100,06
R1_600	601	15691,37	RC2_600	601	5732,08
R1_800	801	27650,67	RC2_800	801	9217,99
R1_1000	1001	42311,91	RC2_1000	1001	13631,76
Summe		211980,24	Summe		163037,46

Tabelle 2.5: Beste bekannte Lösungen der Gehring-Instanzen

Die C2-Instanzen nehmen jedoch eine Sonderstellung ein. Bei genauer Betrachtung der Lösungen von Mester und Bräysy (2007) fällt die Anzahl der benötigten Fahrzeuge auf. Die Autoren bemerken in ihrer Analyse ebenfalls, dass die Distanzen im Rahmen der C2-Instanzen im Vergleich zu den anderen Instanztypen und im Vergleich zu Lösungen, die die Zeitfenster berücksichtigen, sehr viel geringer sind. Weiter gehen die Autoren nicht darauf ein. Allerdings beträgt im Falle der Instanz C2_200 die maximale Kapazität eines Fahrzeuges 700. Die Summe der Nachfragen aller Kunden liegt bei 3770 Einheiten. Daraus ergibt sich nach Toth und Vigo (2002a, S. 490) eine Mindestanzahl von $\frac{3770}{700} \approx 5,39$ Fahrzeugen[54]. Das bedeutet, um alle Kunden zu beliefern, sind mindestens 5,39 Fahrzeuge erforderlich. In den Ergebnissen von Mester und Bräysy (2007) werden bei der Instanz C2_200 jedoch nur 4

[54] Hier wird von der Ganzzahligkeitsbedingung abgesehen.

Instanz	Mester und Bräysy (2007)	Minimale Fahrzeuganzahl
C2_200	4	5,4
C4_400	8	10,8
C4_600	12	16,5
C4_800	16	21,7
C4_1000	20	27,4

Tabelle 2.6: Routenanzahl in C2-Instanzen von Gehring und Homberger (1999)

Fahrzeuge benötigt, wodurch die Ergebnisse in dieser Instanzklasse nicht valide sind. In Tabelle 2.6 finden sich die minimal benötigte Anzahl der Fahrzeuge für die C2-Instanzen sowie die Fahrzeuganzahl, die Mester und Bräysy (2007) benötigt haben. Bei allen fällt auf, dass zu wenige Fahrzeuge verwendet werden, um alle Kunden bedienen zu können. Deshalb werden die C2-Instanzen bei den numerischen Analysen in Kapitel 4.2 ausgeklammert bzw. im Rahmen dieser Arbeit „neue“ beste Lösungen geliefert.

2.6 Literaturüberblick zu Vehicle-Routing-Problemen

In diesem Abschnitt sollen Lösungsverfahren der Literatur für das CVRP näher betrachtet werden. Der Fokus liegt hierbei vorwiegend auf Beiträgen, in denen die Verfahren auf den Instanzen von Golden et al. (1998) getestet werden. In Cordeau et al. (2005) findet sich ein Überblick über Verfahren zur Lösung des CVRPs. Deshalb sollen im Folgenden überwiegend Arbeiten berücksichtigt werden, die 2006 oder später veröffentlicht sind, die nicht in dem Beitrag von Cordeau et al. (2005) Betrachtung finden, jedoch für die vorliegende Arbeit eine große Rolle spielen oder ebenfalls von Cordeau et al. (2005) erwähnt werden, jedoch für die weiteren Ausführungen von großer Bedeutung sind und deshalb nochmals erläutert werden. Es finden sich insbesondere viele Arbeiten, die den evolutionären Algorithmen zugeordnet werden können, da diese einen inhärenten Parallelismus besitzen, der für diese Arbeit ebenfalls bedeutungsvoll ist. Einen Überblick über jüngere Algorithmen zur Lösung des CVRPs findet man weiterhin in den Beiträgen von Laporte (2009) oder Marinakis und Migdalas (2007).

2.6.1 Parallele Algorithmen in der Tourenplanung

An dieser Stelle soll auf parallele Algorithmen eingegangen werden. Der Fokus liegt grundsätzlich auf Verfahren, die zur Lösung des CVRPs genutzt werden. Da es jedoch auch einige Verfahren gibt, die nicht für das CVRP konzipiert sind, aber dennoch interessante Aspekte bzgl. der Parallelisierungsmöglichkeiten aufzeigen, soll sich nicht auf reine CVRP-Verfahren beschränkt werden. Zunächst werden rein GPU-basierte Ansätze näher betrachtet, bevor darauf folgend parallele Ansätze im Allgemeinen vorgestellt werden.

2.6.1.1 GPU-basierte Algorithmen

Es gibt viele Publikationen, die sich mit dem TSP als Untersuchungsgegenstand auseinandersetzen und dabei auf die Rechenleistung der GPU zurückgreifen (vgl. z.B. Janiak et al. (2008)). Einer der wenigen Beiträge, der das CVRP mittels der GPU und der Compute Unified Device Architecture (CUDA) von Nvidia bearbeitet, stammt von Schulz (2013). Er untersucht insbesondere die Auswirkungen auf die Performance, wenn verschiedene Mechanismen der Grafikkarte (wie bspw. die Nutzung verschiedener Speichertypen) zum Einsatz kommen. Die Lösungsgüte, die mittels der Lokalsuche, die die Basis für

die Untersuchungen bildet, berechnet wird, wird vollkommen außer Acht gelassen. Aufbauend auf den „Resource extension functions“ von Irnich (2008a,b) verwendet der Autor einen 2-Opt und 3-Opt (Schulz, 2013, S. 17). Die Grundversion der Lokalsuche evaluiert auf der GPU die Zielfunktion und Verletzungen von Restriktionen, findet den besten Nachbarn und wendet ihn auf die aktuelle Lösung an. Der Autor vergleicht keine Speedups mit einer CPU-Implementierung, sondern nur Laufzeitveränderungen bzgl. dieser Grundversion, die ebenfalls die GPU benutzt. Das heißt, er startet mit dieser Basisversion, optimiert sie und zeigt die Auswirkungen bzw. Erfolge und Misserfolge der Optimierungsbemühungen auf. Neben Auswirkungen der Nutzung verschiedener Speicherarten auf der Grafikkarte werden auch die Auswirkungen der Blockgrößen und viele weitere Parameter beleuchtet. In dem Beitrag finden sich keine Ergebnisse, die auf CVRP-Instanzen erzielt werden, sodass man ihn eher als Leitlinie zur möglichst performanten Implementierung von Lokalsuchen ansehen kann.

Einen etwas anderen Ansatz als Schulz (2013) verfolgen Luong et al. (2013). Die Autoren gehen etwas weiter gefasst an die Thematik der Metaheuristiken auf GPUs heran und nutzen ebenfalls CUDA. Sie erläutern die Vorgehensweise wie man die Lokalsuche parallelisieren und schließlich in einer Metaheuristik einsetzen kann. Allerdings beschreiben die Autoren das Verfahren sehr abstrakt, da sie sich nicht auf ein konkretes Untersuchungsobjekt beziehen, sondern es möglichst allgemeingültig halten, um somit viele Einsatzszenarien (Problemstellungen) abzudecken. Hierbei schlagen sie auch Verfahren bzw. Ansätze vor, um Datenstrukturen auf die GPU zu mappen (z.B. Ergebnisrepräsentation). Als Untersuchungsgegenstand wird unter anderem das TSP herangezogen. Hauptaugenmerk liegt auch hier nicht auf der Lösungsgüte, sondern auf dem Speedup. Dieser wird im Vergleich zu Schulz (2013) gegenüber einer CPU gemessen, wobei sowohl unterschiedliche Grafikkarten als auch unterschiedliche CPUs zum Einsatz kommen. Für kombinatorische Optimierungsprobleme erzielen die Autoren einen Speedup bis zu 50 im Vergleich zur CPU-Version.

Ähnlich wie zuvor beschrieben, erläutern Luong et al. (2011) wie man eine Multistart-Lokalsuche auf der GPU implementieren kann. Der Fokus liegt dabei auf der Umsetzung einer Version bei der die Lokalsuchen parallel und unabhängig voneinander auf der GPU ausgeführt werden. Es besteht jedoch die Möglichkeit, dass die besten gefundenen Lösungen zwischen den Lokalsuchen ausgetauscht werden. Die Autoren verwenden bzw. empfehlen die Verwendung unterschiedlicher Speichertypen auf der Grafikkarte - je nach Problemtyp. Zur Anwendung kommen eine TS und ein SimA und diese werden mittels des Quadratic-Assignment-Problems getestet. Ebenso wie in den bereits vorgestellten GPU-basierten Verfahren, wird auch in diesem Beitrag die Lösungsgüte außer Acht gelassen. Die Speedups werden in Analogie zu Luong et al. (2013) im Vergleich zu einer CPU ermittelt und betragen maximal 12.

Li et al. (2011) schlagen einen parallelen GA vor, der mittels der GPU beschleunigt wird. Das Untersuchungsobjekt ist dabei das MDVRP. Der GA der Autoren beinhaltet wie die meisten anderen Verfahren die Schritte der Selektion, des Crossovers sowie der Mutation. Die Parallelisierung kommt durch die Verteilung der Individuen auf Threads zu Stande, d.h., jedes Chromosom wird einem Thread zugeordnet, der dann die genetischen Operationen darauf anwendet.[55] Zur Ausführung des Crossovers ist ein weiteres Individuum notwendig, das dem Thread zufällig zugeordnet wird. Die Autoren testen mit Instanzen einer Größe von 50 bis 200 Knoten und erreichen einen maximalen Speedup von ca. 960

[55] Die Beschreibung des Verfahrens und auch der Implementierung als solche ist sehr abstrakt gehalten, sodass sie nicht ohne Weiteres nachimplementiert werden kann.

gegenüber der CPU. Hierbei muss jedoch angemerkt werden, dass eine aktuelle GPU mit einer zwei Jahre älteren CPU verglichen wird, was den Aussagegehalt des Speedups schmälert (vgl. Fischer (2010); Intel (o.J.)). Auch in diesem Beitrag wird die Lösungsgüte vollkommen außer Acht gelassen.

Coelho et al. (2012) stellen einen GPU-basierten Ansatz für das Single-VRP with Deliveries and Selective Pickups vor. Das Verfahren basiert auf einer VNS bei der die Nachbarschaften Swap-, 2-Opt-, Relocate- und Or-Opt-Operator zum Einsatz kommen. Die Parallelisierung kommt bei der Evaluierung der Nachbarschaft zu Stande. Das heißt, jede Nachbarschaft wird auf der GPU vollständig betrachtet. Für jeden potentiellen Move wird ein Wert in die Ergebnismatrix geschrieben. Die so erhaltene Matrix wird im Ganzen zurück zur CPU kopiert und diese sucht daraus den besten Move. Es zeigt sich hier, dass ein sehr hoher Aufwand zum Kopieren der Daten von der GPU zur CPU zu Stande kommt. Die Autoren geben neben den Speedups, die bis zu etwa 43 betragen, auch die Lösungsqualität an, was im Kontrast zu den bisher vorgestellten Algorithmen steht, die vorwiegend nur auf das methodische Vorgehen abgestellt haben.

In dem Beitrag von Nashed et al. (2012) werden drei Metaheuristiken auf ihre Tauglichkeit bzw. auf die Performanceauswirkungen bei Unterstützung durch die GPU untersucht. Es handelt sich um Particle Swarm Optimization, Differential Evolution und Scatter Search. Die Autoren verwenden für ihre numerische Studien rechenintensive mathematische Funktionen, sodass die Implementierungen nicht ohne Weiteres auf kombinatorische Optimierungsprobleme übertragbar sind. Das Ziel der Autoren ist dabei weniger einen Speedup zu erlangen, sondern vielmehr genauere Ergebnisse für die mathematischen Funktionen in gleichbleibender Zeit zu erzielen, was ihnen in vielen Fällen gelingt.

Weyland et al. (2013) stellen eine Metaheuristik vor, die auf der Monte-Carlo-Methode basiert und am Beispiel des Probabilistic-TSP with Deadlines untersucht wird. Die GPU übernimmt die Aufgaben der Evaluierung der Zielfunktion sowie der Verletzung der Restriktionen.

Weitere Parallelisierungsansätze von Metaheuristiken werden bspw. in der Arbeit von Luong et al. (2012) diskutiert. Allgemein lässt sich feststellen, dass es eine Vielzahl von GPU-basierten Algorithmen gibt, die auf Metaheuristiken aufbauen, die inhärent parallel sind, wohingegen es weniger GPU-basierte Verfahren gibt, die sich mit sequentiellen Algorithmen beschäftigen (Luong et al., 2012, S. 369f.).

Es zeigt sich an dem Mangel an Literatur über Verfahren zur Lösung des CVRP durch GPU-Unterstützung, dass in diesem Forschungsfeld noch viele Fragen offen sind und diese Arbeit einige dieser Fragen beantworten wird. Insbesondere dem Verzicht auf einen Vergleich der Lösungsgüte mit State-of-the-Art-Algorithmen soll in dieser Arbeit begegnet werden.

2.6.1.2 Nicht-GPU-basierte Algorithmen

Es werden auch nicht-GPU-basierte parallele Algorithmen betrachtet, da sich manche Implementierungsdetails nicht von einer GPU-Implementierung unterscheiden und in dieser Arbeit zum Einsatz kommen.

Jin et al. (2012) entwickeln eine parallele TS zur Lösung des CVRPs. Der Algorithmus beginnt mit der Generierung einer Startlösung, die danach von mehreren Suchthreads verbessert wird. Ein Suchthread enthält jeweils eine TS, die jedoch auf verschiedenen Nachbarschaften aufbaut. Die Threads kommunizieren zu bestimmten Zeitpunkten über einen Lösungspool. Das heißt, sobald eine TS beendet ist, wird die beste gefundene Lösung an den Lösungspool gesendet. Nachdem alle Lösungen eingesammelt sind, entscheidet ein Masterprozessor welche Lösung in der nächsten Iteration als Startlösung für alle

Suchthreads dienen soll. Jin et al. (2012) bauen auf der granularen TS von Toth und Vigo (2003), die auch in dieser Arbeit verwendet wird, auf, bei der die Suche auf vielversprechende Kanten beschränkt wird. Als Nachbarschaften verwenden die Autoren den Relocate-, Swap- und 2-Opt*-Operator. Weiterhin sind zwischenzeitlich ungültige Lösungen erlaubt, wobei mit einer Strafkostenfunktion gearbeitet wird. Insgesamt kommen vier TS zum Einsatz. Eine Besonderheit besteht darin, dass eine TS nur für die Diversifizierung abgestellt ist, d.h. die Wahrscheinlichkeit, dass dieser Thread neue beste Lösungen des Suchprozesses liefert, ist nur sehr gering, stattdessen werden sich sehr stark unterscheidende Lösungen generiert.

Ein ähnliches Verfahren stammt von Jin et al. (2011) bei dem auch eine parallele TS zum Einsatz kommt. Auch hier wird auf der granularen TS von Toth und Vigo (2003) aufgebaut. Weiterhin werden mehrere TS-Threads gestartet, die mit jeweils unterschiedlichen Nachbarschaften arbeiten. Zu diesen zählen der Relocate-, Swap-, 2-Opt*- und Cross-Exchange-Operator. Letzterer ist dabei beschränkt auf Kundensequenzen der Länge von 1-3, die ausgetauscht werden.

Ebenfalls eine parallele TS wird von Cordeau und Maischberger (2012) vorgeschlagen. Der Ansatz dient nicht nur zur Lösung des CVRPs, sondern auch des PVRPs, MDVRPs und Site-Dependent-Vehicle-Routing-Problems. Er ist also auf eine breite Zahl von Problemstellungen anwendbar und somit sehr flexibel. Die Autoren verwenden die TS als Lokalsuche, die in einer Iterated Local Search eingebettet ist und auf der Arbeit von Cordeau et al. (1997) aufbaut. Weiterhin sind zwischenzeitlich ungültige Lösungen erlaubt und es wird mit einer Strafkostenfunktion gearbeitet. Parallelisiert wird der Algorithmus dadurch, dass die Suchthreads mit unterschiedlichen Lösungen gestartet werden und darauf die TS angewendet wird. Weiterhin wählt jeder Suchthread die Parameter zur Steuerung der TS individuell, wodurch eine größere Diversifizierung zwischen den Suchprozessen erreicht werden kann. Die Kommunikation zwischen den Prozessen wird mittels eines Crossovers erreicht. Jeder Suchthread wählt einen anderen Crossover-Partner und führt den Crossover schließlich aus.

Ein weiterer paralleler Algorithmus zur Lösung des CVRPs stammt von Groër et al. (2011). Das Besondere an dem Verfahren liegt in der Verknüpfung von heuristischen mit exakten Methoden. Konkret setzt sich das Verfahren aus einem Set-Covering-Solver und der Metaheuristik RRTR zusammen. Der RRTR wendet eine Lokalsuche an, wobei sowohl verbessernde als auch einige verschlechternde Moves akzeptiert werden. Als Nachbarschaftsstrukturen kommen der Or-Opt-, 3-Opt- sowie der 3-Point-Move[56] zum Einsatz. Zur Diversifizierung der Suche während des RRTR werden einige Kunden aus der aktuellen Lösung entnommen und danach wieder an anderen Positionen eingesetzt. Der Set-Covering-Solver setzt Lösungen bzw. einzelne Routen wieder zu einer neuen Lösung zusammen. Die Kommunikation der Komponenten findet über einen Masterprozessor statt. Dieser nimmt Lösungen entgegen und wählt Lösungen bzw. Routen zur weiteren Bearbeitung aus und sendet sie an die heuristischen Komponenten und den Set-Covering-Solver. Außerdem ist der Master-Prozessor dafür zuständig, die Problemstellung weiter zu zerlegen. Dabei werden Routen, die in vielen unterschiedlichen Lösungen enthalten sind, aus der Lösung entfernt, da man annimmt, dass Routen, die oftmals vorkommen, Teil der besten Lösung sind und damit nicht weiter beachtet werden müssen. Das bedeutet, die Knoten, die auf diesen Routen liegen, werden aus der Problemstellung entfernt und es muss nur noch mit weniger Problemdaten gearbeitet werden. Getestet wird der Algorithmus mit bis zu 129 Prozessoren, von denen maximal 16

[56] Der 3-Point-Move vertauscht die Position von zwei zusammenhängenden Kunden mit einem anderen Kunden (Groër et al., 2011, S. 319). Man könnte den Operator demnach als 1-2-Cross-Exchange ansehen.

Set-Covering-Solver und der Rest RRTR-Solver sind.

Barbucha (2011) schlägt einen Algorithmus zum Lösen des CVRPs vor, der auf kooperierenden Agenten basiert. Dabei gibt es einen Master, der die Agenten steuert. Die Steuerung erfolgt über die Zuteilung von Lösungen an die Agenten bzw. über das Einsammeln von Lösungen der Agenten. Letztgenannte versuchen parallel eine Lösung des CVRPs, das sie zugeteilt bekommen, zu optimieren. Der Master verwaltet eine Population von Lösungen und liefert zum Ende des Algorithmus die beste gefundene Lösung.

Eine der frühen Arbeiten, die sich mit der Parallelisierung von Lösungsverfahren des CVRPs beschäftigt, stammt von Taillard (1993). In dem Beitrag wird die Parallelisierung durch eine Problemdekomposition erreicht, d.h., Kunden werden in Regionen eingeteilt und diese werden parallel behandelt. Dabei wird eine TS eingesetzt, die als Nachbarschaft den Swap-Operator verwendet und während des gesamten Suchprozesses nur gültige Lösungen akzeptiert.

Doerner et al. (2004) untersuchen in ihrer Arbeit verschiedene Implementierungen von Ant-Colony-Optimierung zur Anwendung auf CVRPs. Die Autoren implementieren ein Rank-based-Ant-System, ein Max-Min-Ant-System und ein Ant-Colony-System, die sich zumeist nur hinsichtlich der Berechnung des Pheromonupdates unterscheiden. Schließlich parallelisieren die Verfasser das Rank-based-Ant-System und zeigen die Effekte der Parallelisierung auf. Sie gehen jedoch nicht auf qualitative Ergebnisse bzgl. Benchmarkinstanzen ein.

Ein paralleles Lösungsverfahren für das VRPTW stammt von Garcia et al. (1994). Die Autoren verwenden dabei eine TS. Die Parallelisierung erfolgt durch Aufteilung der Knoten in einer Probleminstanz auf die beteiligten Prozessoren. Verwendet werden der 2-Opt*- und Or-Opt-Move. Anhand des Knotens wird der Move determiniert, welcher bewertet werden muss.

Weitere parallele Verfahren zur Lösung des CVRPs stammen bspw. von Jozefowiez et al. (2002) oder Ralphs et al. (2003). In dem Beitrag von Alba et al. (2012) wird sich ausführlich mit parallelen Metaheuristiken, der Hardware, die zur Parallelisierung verwendet wird, und auch mit den Programmiersprachen beschäftigt. Deshalb wird an dieser Stelle nicht weiter auf parallele Metaheuristiken eingegangen, sondern an diesen Beitrag verwiesen. Weitere parallele Algorithmen im Kontext des VRP werden in der Arbeit von Crainic (2008) vorgestellt.

2.6.2 Evolutionäre Algorithmen in der Tourenplanung

Da GAs bzw. evolutionäre Algorithmen einen inhärenten Parallelismus besitzen, sollen im Folgenden einige GAs vorgestellt werden, die zwar nicht explizit als paralleler Ansatz veröffentlicht, aber durch den inhärenten Parallelismus leicht zu parallelisieren sind. Der Fokus liegt auf Verfahren, die für das CVRP vorgeschlagen werden bzw. Ergebnisse für CVRP-Benchmarkinstanzen liefern.

Pereira et al. (2002) schlagen einen reinen GA zur Lösung des CVRPs vor. Das Besondere an ihrer Arbeit ist der neuartige Crossover mit Hilfe dessen eine Subroute von einem Individuum an das andere Elternteil vererbt wird. Es kommt eine Datenstruktur zum Einsatz, die die einzelnen Routen strikt voneinander trennt und so Operationen auf Routenbasis einfach ermöglicht. Als Mutationsoperatoren verwenden die Autoren einen Swap-, Inversion-, Insertion- und Displacement-Operator. Getestet wird der Algorithmus mit vergleichsweise kleinen Instanzen, auf denen jedoch gute bis sehr gute Ergebnisse erzielt werden.

In dem Verfahren von Berger und Barkaoui (2003) kommt ebenfalls ein hybrider Ansatz zum Einsatz.

Der Algorithmus verwaltet zwei unterschiedliche Populationen, zwischen denen Individuen zu definierten Zeitpunkten migrieren. Bei der Selektion wird die klassische Roulette-Wheel-Selektion eingesetzt. Der Crossover zweier Individuen nutzt viele Informationen über die Struktur der beiden. Es wird das aktuelle Routing von Knoten auf Routenebene berücksichtigt, um damit vielversprechende Elemente der Elternteile zu erhalten. Darauf folgend kommen Mutationsoperatoren zum Einsatz, die auf Lokalsuchen basieren und eine Large Neighborhood Search.

Alba und Dorronsoro (2004) nutzen zur Lösung des CVRPs einen „Cellular Genetic Algorithm“. Das Besondere an dem Verfahren ist das Management der Population, deren Individuen in einem 2D-Raum angeordnet sind. Aufgrund dieser Anordnung ist es möglich, direkte Nachbarn eines Chromosoms zu bestimmen. Diese Fähigkeit macht sich der Algorithmus zu Nutze, damit sich nur benachbarte Individuen paaren. Das CVRP wird als große Tour mit Tourseparatoren verwaltet. Als Rekombinationsoperator findet der edge recombination operator Anwendung und als Mutationsoperatoren werden Insertion-, Swap- und Inversion-Operatoren verwendet. Weiterhin wird eine Lokalsuche bestehend aus einem 2-Opt- und einem λ-Interchange-Operator auf die erzeugten Individuen angewendet. Die Autoren erlauben ungültige Lösungen, indem sie diese mit höheren Kosten bestrafen. Erzeugte Individuen ersetzen ihre Eltern, wenn Sie eine bessere Fitness als diese aufweisen. Alba und Dorronsoro (2008) und Alba und Dorronsoro (2006) präsentieren einen ähnlichen Algorithmus mit umfangreichen Ergebnissen zu verschiedensten Benchmarkinstanzen.

Ein weiterer hybrider evolutionärer Algorithmus stammt von Prins (2004), bei dem der OX, der insbesondere seinen Einsatz beim TSP findet, zur Anwendung kommt. Als Mutationsoperator wird eine Lokalsuche eingesetzt. Der OX wird von dem Autor auf die gleiche Art und Weise wie bei Betrachtung eines TSP genutzt, was dadurch ermöglicht wird, dass das vorgeschlagene Verfahren mit einer Datenstruktur, die auf einer großen Tour basiert, arbeitet, jedoch auf Tourseparatoren verzichtet. Für jede so generierte Permutation existiert eine kostenminimale große Tour mit Tourseparatoren. Zur Bestimmung dieser kostenminimalen Tour, d.h., an welche Stelle die Tourseparatoren am besten gesetzt werden sollten, schlägt der Autor ebenfalls einen effizienten Algorithmus vor. Als Mutationsoperatoren kommen neun verschiedene Lokalsuchen wie bspw. der Relocate- oder Swap-Operator zum Einsatz. Ein weiterer wichtiger Bestandteil des Verfahrens ist das Vermeiden von doppelten Lösungen/Individuen innerhalb einer Population (Klone). Als Entscheidungskriterium wird nicht auf die vollständige Lösung abgestellt, sondern lediglich auf die Zielfunktion der Lösungen. Das heißt, wenn von zwei unterschiedlichen Lösungen die Zielfunktionswerte fast identisch sein sollten, wird eine der beiden Lösungen nicht in die Population aufgenommen werden, da angenommen wird, dass es sich bei den Lösungen um Klone handelt.

Dorronsoro et al. (2007) bauen auf dem Algorithmus von Alba und Dorronsoro (2004) auf und erweitern ihn um die Fähigkeit, mehrere Populationen zu organisieren. Die Populationen werden jeweils auf einem Rechner verwaltet und die Population an sich folgt dem „cellular model“ (Dorronsoro et al., 2007, S. 2). Im Unterschied zu dem ursprünglichen Algorithmus von Alba und Dorronsoro (2004) sind keine ungültigen Lösungen mehr erlaubt; sobald eine ungültige Lösung auftritt, wird sie einem Reparaturmechanismus unterzogen, um wieder eine gültige herzustellen. Die Autoren verwenden bei der Rekombination den Operator, der von Pereira et al. (2002) vorgeschlagen wird, um eine frühzeitige Konvergenz zu einem lokalen Optimum zu vermeiden. Die Mutationsphase wird im Vergleich zu Alba und Dorronsoro (2004) um einen weiteren Operator - der Dispersion - erweitert. Schließlich endet die

Generierung eines Individuums mit einer Lokalsuche, die aus einem 1-Interchange und 2-Opt besteht. Das Verfahren wird auf einem Servercluster getestet.

Nagata und Bräysy (2009) verfolgen ebenfalls einen hybriden genetischen Ansatz. Sie adaptieren den sogenannten edge assembly crossover, der ursprünglich für das TSP entwickelt wurde, auf das CVRP. Die Grundlage der Publikation bildet die Arbeit von Nagata (2007). Dabei lassen sie temporär ungültige Lösungen zu, die jedoch in einem weiteren Schritt einer Modifikationsphase unterzogen werden, um wieder eine gültige Lösung zu erhalten. Dazu verwenden die Autoren eine Strafkostenfunktion. Im Anschluss wird jede Lösung einer Lokalsuche unterzogen, bevor die so erzeugte Lösung die Elternlösung, sofern sie eine geringere Distanz aufweist, ersetzt. Der Kernpunkt bei diesem Verfahren liegt in der Crossover-Funktion, die sehr komplex gestaltet ist.

Ein evolutionärer Ansatz wird von Mester et al. (2007) verwendet, wobei er sich erheblich von anderen populationsbasierten Heuristiken unterscheidet. Die Autoren stellen ihren Algorithmus als eine evolutionäre Strategie vor, bei der die Größe der Population bei eins liegt und pro Generation nur ein Nachkomme erzeugt wird. Die aktuelle Lösung wird einem Mutationsoperator unterzogen, der viele verschiedene Lösungen (Nachkommen) erzeugen kann. Im Rahmen der Mutation wird eine definierte Anzahl von Kunden entfernt, die danach wieder eingefügt wird. Eine Lokalsuche, die periodisch während dem Einfügen mit einer bestimmten Wahrscheinlichkeit sowie am Ende, wenn alle Kunden wieder eingefügt sind, ausgeführt wird, begleitet den Prozess. Weiterhin verwenden die Autoren in ihrer Lokalsuche eine Beschränkung der Nachbarschaft. Diese wird anhand einer Einteilung des 2D-Raumes auf Basis des Depotstandorts durchgeführt.

Einen ähnlichen Ansatz wie Mester et al. (2007) verfolgen Mester und Bräysy (2007). Bei diesem Verfahren wird die evolutionäre Strategie von Mester et al. (2007) als Bestandteil einer Phase des Algorithmus eingesetzt. In der ersten Phase verwenden die Autoren eine sogenannte „penalty variable neighborhood“, die mit Hilfe einer Guided-Local-Search durchsucht wird. Die Methode versucht, den Suchraum auf Routen, die nah beieinander liegen, einzuschränken. Das Verfahren wird auf unterschiedlichen Benchmarkinstanzen getestet. Dabei kommen bspw. die Instanzen von Gehring und Homberger (1999), die für das VRPTW vorgeschlagen werden, zum Einsatz, indem die Zeitfensterbeschränkung relaxiert wird.

Vidal et al. (2011) stellen ebenfalls einen hybriden GA für das MDVRP, PVRP und Multi-Depot-Periodic-VRP vor. Das Verfahren generiert aus zwei Elternlösungen Nachwuchs, der dann „unterrichtet“ wird. Die Autoren verwenden einen neuen sog. periodic crossover with insertions, der explizit auf PVRPs zugeschnitten ist. Der „Unterricht“ findet in Form einer Lokalsuche, die an Stelle des Mutationsoperator im GA zum Einsatz kommt, statt. Diese beinhaltet neben Route- und Pattern-Improvement-Methoden einen Reparaturmechanismus, um ungültige Lösungen zu gültigen zu transformieren. Die Autoren testen ihren Algorithmus auch mit CVRP-Benchmarks.

2.7 Hardwarearchitektur von Grafikkarten

Bisher stehen Algorithmen zur Lösung des CVRPs im Vordergrund der Betrachtungen. Da in dieser Arbeit jedoch auch die verwendete Hardware - sprich GPU - Hauptuntersuchungsgegenstand ist, soll nun auf die Eigenschaften der Hardware im Allgemeinen und die der GPU im Speziellen eingegangen werden.

Obwohl es viele Bestrebungen gibt, die Taxonomie von Flynn (1972) abzulösen (vgl. bspw. Duncan (1990)) soll in Anlehnung an die Taxonomie in diesem Abschnitt auf grundlegende Eigenschaften von Hardwarearchitekturen eingegangen werden. Flynn (1972, S. 949) unterscheidet zwischen „single-instruction stream single-data stream organization (SISD)“, „single-instruction stream-multiple-data stream (SIMD)“, „multiple-instruction stream-single-data stream organizations (MISD)“ sowie „multiple-instruction stream-multiple-data stream (MIMD)“. Grafikkarten sind der SIMD-Architektur zuzuordnen, da der gleiche Befehl auf viele verschiedene Daten angewendet wird.[57] Aufgrund dessen wird im Folgenden der Fokus auf der SIMD-Architektur liegen. Zunächst findet sich ein kurzer Literaturüberblick über Arbeiten[58], die sich mit Hardwarearchitekturen von Multicore-Prozessoren beschäftigen. Danach wird die Entwicklung der GPUs kurz skizziert, bevor letztendlich auf die Hardwarearchitektur der Grafikkarte, die der Arbeit zugrunde liegt, und auf das Programmiermodell CUDA eingegangen wird.

2.7.1 Literaturüberblick zu Surveys von Multicore-Prozessoren

Varbanescu (2010, S. 9-33) gibt in seiner Arbeit einen Überblick über verschiedenste aktuelle Multicore-Prozessortypen. Der Beitrag bildet einen guten Ausgangspunkt, um sich mit den unterschiedlichen Typen vertraut zu machen. Der Autor betrachtet Embedded-Multicore-Prozessoren und Multicore-Prozessoren, wie sie in üblichen Desktop-Rechnern und Servern verwendet werden, sowie Grafikkarten und Prozessoren, die er in das Feld des High-Performance Computings (HPC) einordnet. Diese Kategorisierung lässt zwar Fragen offen, da in das Feld des HPCs bspw. der Cell/B.E., der unter anderem in der Playstation 3 zum Einsatz kommt, genannt wird. Trotzdem wird dem Leser ein guter Einblick in die verschiedenen Hardwarearchitekturen ermöglicht. Varbanescu (2010, S. 27-32) schließt seine Arbeit, indem er die Architekturen untereinander vergleicht und einander tabellarisch gegenüberstellt.

Buchty et al. (2009) beschreiben ebenfalls moderne Multicore-Prozessoren. Dazu gehen sie zunächst auf allgemeine Eigenschaften, die Prozessoren aufweisen können, ein. Darunter wird bspw. auf die verschiedenen Möglichkeiten, ein Programm zu parallelisieren, eingegangen. Darauf folgend beleuchten die Autoren einige wenige Multicore-Prozessoren, bevor sie sich dann den Programmiermodellen, die den unterschiedlichen Architekturen zu Grunde liegen, widmen. Außerdem liefern die Verfasser noch einige Beispiele für Anwendungen, die von Multicore-Prozessoren Gebrauch machen.

In der Arbeit von Sodan et al. (2010) werden Differenzierungskriterien für Multicore-Prozessoren vorgestellt, d.h., es wird implizit versucht, die Taxonomie von Flynn (1972) zu erweitern. Die Differenzierungskriterien werden schließlich dazu genutzt, um verschiedenste Prozessoren tabellarisch einander gegenüberzustellen. Weiterhin gehen die Verfasser kurz auf die Anforderungen an die Softwareentwicklung, die durch die Parallelisierung entstehen, ein.

Ein weiterer Artikel, der sich mit der Parallelisierung bzw. der Hardware, auf der die Parallelisierung umgesetzt wird, beschäftigt, stammt von Brodtkorb et al. (2010). Die Autoren beschreiben unterschiedliche Arten der Parallelisierung, die je nach Hardware zum Einsatz kommen. Sie unterscheiden zwischen „Multi-chip parallelism“, „Multi-core parallelism“, „Multi-context (thread) parallelism“ und „instruction parallelism“ (Brodtkorb et al., 2010, S. 3). Weiterhin gehen die Autoren auf verschiedene Speichertypen,

[57] Wenn man die Taxonomie von Flynn (1972) erweitert, kann man GPUs nach McCool (2008, S. 821) in die Kategorie „single-program multiple-data“ (SPMD) einordnen, die eine Kombination aus SIMD und MIMD darstellt.

[58] Des Fokus liegt hier vornehmlich auf Surveys, da sie einen guten Einstieg in die Thematik der Multicore-Prozessoren geben.

die moderne Hardware-Architekturen bieten, ein. Auch hier beleuchten die Verfasser verschiedene Prozessoren und deren Eigenschaften. Bei einem Vergleich der unterschiedlichen Prozessortypen werden auch Kosten berücksichtigt, die die Autoren mittels MFlop/s pro Dollar angeben (Brodtkorb et al., 2010, S. 10). Schließlich werden noch diverse Programmiermodelle sowie konkrete Anwendungen vorgestellt.

Rauber und Rünger (2010) sowie Blake et al. (2009) gehen ebenfalls auf die Eigenschaften von Multicore-Prozessoren ein. Dabei finden sich detaillierte Beschreibungen zu möglichen Differenzierungskriterien und es werden Multicore-Prozessoren vorgestellt. Auch die Arbeit von McCool (2008) ist ähnlich aufgebaut wie die bereits genannten. Bemerkenswert ist jedoch die Differenzierung von SIMD und SPMD, die von dem Verfasser vorgenommen wird.

Die zuvor erwähnten jüngeren Artikel haben gemein, dass in allen GPUs ihre Erwähnung finden, was nochmals die wachsende Bedeutung von GPUs als Prozessoren, die nicht zur Darstellung von Grafiken gedacht sind, unterstreicht.

2.7.2 Entwicklungsgeschichte der Grafikkarten

An dieser Stelle soll kurz auf die Entwicklungsgeschichte von Grafikkarten im Allgemeinen eingegangen werden. Ausführlichere Beschreibungen finden sich in den Arbeiten von Borgo und Brodlie (2009) und McClanahan (2010). Ein geschichtlicher Überblick aus Sicht der Hardware von Nvidia wird von Nickolls und Dally (2010) vorgenommen. Dabei beginnen die Autoren ihren Rückblick mit Riva TNT und Geforce 256 im Jahre 1997 bzw. 1999.

Eine der ersten GPUs[59] für Rechner war der Professional Graphics Controller (PGA) von IBM Anfang der 1980er Jahre, wodurch durch die Entkoppelung von CPU und GPU der Grundstein für die weitere Entwicklung von GPUs gelegt wurde (McClanahan, 2010, S. 2). Im Verlauf der Zeit nahmen die GPUs den CPUs, die zuvor alleine für die Grafikausgabe zuständig waren, immer mehr Arbeit ab. 2001 war es erstmals möglich, kleine Programme von der CPU an die GPU zu senden, die dann allein auf der Grafikkarte berechnet werden konnten (McClanahan, 2010, S. 3). Einige der ältesten Arbeiten aus dem Bereich des GPGPU stammen von Proudfoot et al. (2001), Thompson et al. (2002) und Bolz et al. (2003). Ein weiterer Meilenstein war die G80-Architektur von Nvidia, bei der dem Programmierer erstmals ein vollständig programmierbarer Streaming Multiprozessor (SM) (McClanahan, 2010, S. 4) zur Verfügung stand. Mit der Einführung der Fermi-Architektur 2009 und der Kepler-Architektur 2012 hat Nvidia den Trend in seiner Produktsparte bzgl. GPGPU weiter verstärkt. Auch AMD/ATI versucht, die eigenen Grafikkarten für allgemeine Berechnungen zu öffnen, wobei das bei weitem nicht so gut gelingt wie bei Nvidia. Ein Grund hierfür könnte sein, dass die Unterstützung der Nutzer von Nvidia durch das CUDA-Framework sehr viel besser ist[60] als die des AMD/ATI Stream-SDK. Trotz der aufstrebenden neuen Möglichkeiten, die die Grafikkarten in jüngerer Zeit ermöglichen, kann man mit Nickolls und Dally (2010, S. 56) übereinstimmen, dass die Leistungssteigerung in dem Hardwaresegment insbesondere durch Computerspiele, die immer größere Anforderungen an die GPUs stellen, herbeigeführt wurde.

Beide großen Hersteller von GPUs AMD/ATI und Nvidia unterstützen auch den offenen Standard OpenCL, allerdings wird bspw. von Karimi et al. (2010) oder Du et al. (2012) ein Vergleich zwischen CUDA und OpenCL auf Grafikkarten von Nvidia durchgeführt und diesen gewinnt eindeutig das CUDA-

[59] Nach McClanahan (2010) wurde der Begriff „GPU“ erstmalig 1999 verwendet, d.h. lange Zeit, nachdem es bereits Grafikkarten für Rechner gab.

[60] Insbesondere im Sinne der Dokumentation des Frameworks (Fischer und Stiller, 2013, S. 77).

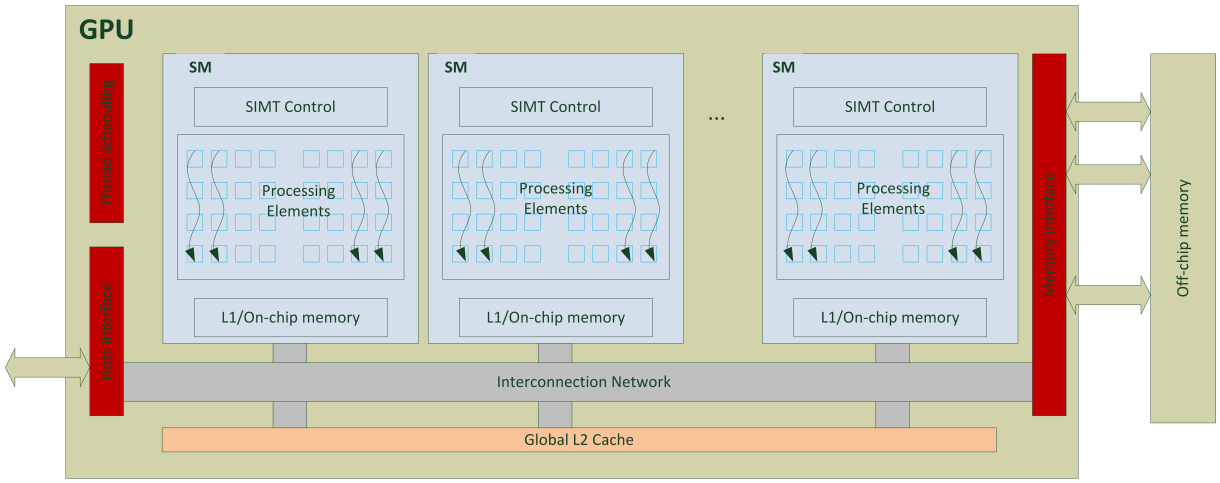


Abbildung 2.25: GPU-Architektur in Anlehnung an Garland und Kirk (2010, S. 62)

Framework. Wenn man die theoretische Geschwindigkeit zwischen AMD/ATI- und Nvidia-Karten betrachtet, stellt man fest, dass erstere eine sehr viel höhere Leistung erbringen können. Aufgrund der besseren Unterstützung von Nvidia, was Dokumentation usw. angeht, finden sich jedoch nur sehr wenige Publikationen, die die Grafikkarten von AMD/ATI zur allgemeinen Berechnung verwenden.

2.7.3 Aufbau von Grafikkarten

Die Hardware - eine Geforce GTX 580 -, die in dieser Arbeit zur Anwendung kommt, stammt von Nvidia. Deshalb wird sich im Folgenden auf die Details der Fermi-Architektur von Nvidia bezogen. Der konzeptionelle Aufbau der Grafikkarten von Nvidia und AMD/ATI ist sich jedoch sehr ähnlich. Die Beschreibung soll lediglich grundlegende Kenntnisse bzgl. der Hardwarearchitektur vermitteln, wobei der Fokus auf dem CUDA-Framework liegt. Dazu wird zunächst auf die Eigenschaften der Prozessoren, die in GPUs verbaut sind, eingegangen, bevor darauf folgend auf die Merkmale des GPU-Speichers eingegangen wird.

Detailliertere Beschreibungen findet man bspw. in dem Beitrag von Nickolls und Kirk (2011), in dem die Tesla-Architektur, die Vorgängerin der hier betrachteten Fermi-Architektur, erläutert wird. Einen Grobüberblick über Nvidias neuere Hardwaregeneration Fermi liefert Wittenbrink et al. (2011), wobei der Fokus der Arbeit sehr stark auf graphischer Darstellung liegt. Nickolls und Dally (2010) hingegen beschreiben im Detail die Fermi-Architektur aus CUDA-Sicht. Die Autoren gehen dabei sowohl auf die Speicherhierarchie als auch auf die Prozessoren ein.

Abbildung 2.25 zeigt die Fermi-Architektur anhand derer der Aufbau der Prozessoren von GPUs von Nvidia beschrieben werden soll. Die GPU besteht aus mehreren Multiprozessoren (multithreaded SMs), die sich wiederum aus mehreren Streaming Prozessor Kernen (SP) zusammensetzen (Nickolls und Kirk, 2011, A.1 S.10). Ein Block[61] wird von genau einem SM ausgeführt bzw. an diesen gebunden und dessen eigentliche Threads wiederum von den SPs bearbeitet. Das Thread Scheduling übernimmt die Aufgabe, die Blöcke auf die SMs zu verteilen, wohingegen jeder SM die Threads seiner zugewiesenen Blöcke selbst mittels des Warp Schedulers verteilt (SIMT Control), was ohne Zeitverlust (zero-overhead) vonstatten geht (Nickolls und Dally, 2010, S. 61-62). Jeder SP besitzt eine eigene Floating-Point-

[61] Auf das Konzept der Blöcke wird in Abschnitt 2.7.4 näher eingegangen.

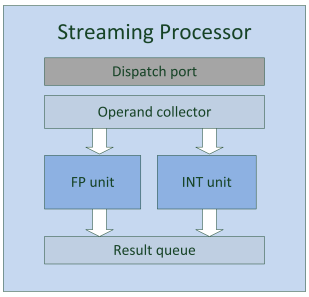


Abbildung 2.26: Streaming Processor in Anlehnung an Wittenbrink et al. (2011, S. 53)

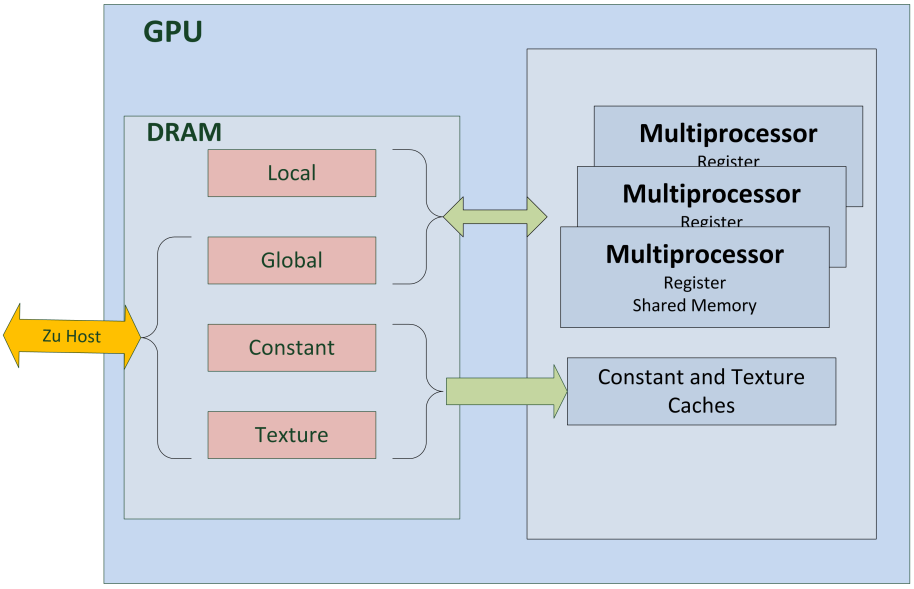


Abbildung 2.27: Speicherhierarchie nach Nvidia (2011a, S. 24)

(FP) und Integer-Einheit (INT) (vgl. Abbildung 2.26). Damit ist es möglich, in einem SM bis zu 32 arithmetische Operationen pro clock cycle durchzuführen (Nickolls und Dally, 2010, S. 63)[62].

Damit die einzelnen Threads und Blöcke miteinander kommunizieren können, sind die Speicher, auf die sie zugreifen können, von großer Bedeutung. Da der Speicherzugriff heute als der eigentliche Flaschenhals angesehen werden kann (vgl. McCool (2008, S. 821), Brodtkorb et al. (2010, S. 3)), ist es wichtig, die Speicherhierarchie, die auf jeder Ebene verschiedene Vor- und Nachteile bietet, zu verstehen.[63] Bei den GPUs von Nvidia gibt es im Gesamten vier Speicherarten, auf die mit Hilfe des CUDA-Frameworks vom Entwickler zugegriffen werden kann. Abbildung 2.27 zeigt die Speicherarten. Zu nennen sind der Off-chip Speicher oder auch Globalspeicher (Global), der L2-Cache (siehe dazu Abbildung 2.25), der L1-Cache bzw. On-chip Memory oder auch Shared Memory (SMem) sowie Speicher, auf die jeder Thread nur für sich allein zugreifen kann (Local). Auf die exakte Funktionsweise soll hier nicht weiter eingegangen werden, da sie zum Verständnis der Implementierung nicht benötigt wird.

[62] Es sei hier schon vorweggegriffen, dass damit nicht ein Warp der Größe 32 in einem Taktzyklus ausgeführt wird. Ein SM besitzt jeweils zwei Warp Scheduler, von denen beide jeweils einen Warp schedulen. Beide Einheiten schedulen dann eine Instruktion von einem Warp für 16 Kerne im SM (Nvidia, 2009, S. 10). Das heißt um eine Instruktion für einen Warp à 32 Threads abzuschließen (sodass die Instruktion auf allen 32 SP durchgelaufen ist), sind insgesamt zwei Taktzyklen nötig.

[63] Durch Berücksichtigung der verschiedenen Speicherarten und deren entsprechende Nutzung, können erhebliche Performancegewinne bzw. -verluste, wie man in Abschnitt 4.1.1.1 lesen kann, erzielt werden.

Details zu den beiden Caches können im Beitrag von Nickolls und Dally (2010, S. 62) nachgelesen werden. Der Globalspeicher ist für alle SMs bzw. SPs sowohl lesend als auch schreibend zugreifbar. Die Größe entspricht der des auf der Grafikkarte verbauten Speichers[64]. Weiterhin kann dieser Speicher von der CPU über die PCI-Express-Schnittstellen beschrieben und ausgelesen werden, d.h., hierüber erfolgt die Kommunikation zwischen CPU und GPU. Allerdings sollte von Datentransfers zwischen GPU und CPU möglichst selten Gebrauch gemacht werden, da mit zunehmender Leistungsfähigkeit der GPU die PCI-Express-Schnittstelle den Engpass darstellt (Gregg und Hazelwood, 2011, S. 136). Demgegenüber steht das SMem, das nur von der GPU heraus angesprochen werden kann. Allerdings ist dieser Teil des Speichers auf jedem einzelnen SM gekapselt, sodass es für einen SM nicht möglich ist, auf das SMem eines anderen SM zuzugreifen. Ein großer Vorteil des SMem ist jedoch, dass die Zeitverzögerung (latency) beim Zugriff im Vergleich zum Globalspeicher bis zu 100 mal geringer ist (Nvidia, 2011a, S. 31). Trotzdem kann der On-chip Speicher zur Synchronisation innerhalb der SPs eines SMs verwendet werden. Eine weitere Speicherart, die auf der GPU verbaut ist, ist der Texturspeicher. Der Speicher ist auf der gleichen Ebene wie der globale Speicher, mit dem Unterschied, dass er gecached ist. Das bedeutet, wenn ein Datum bereits im Texturcache liegt, so kostet der Zugriff auf dieses Datum lediglich die Cachezugriffszeit, andernfalls steigt die Zugriffszeit entsprechend an (Nvidia, 2012, S. 73). Es stellt sich hier berechtigterweise die Frage, wo der Vorteil des Texturspeichers gegenüber dem Globalspeicher, der in neueren Architekturen ebenfalls gecached ist, liegt. Der größte Vorteil dürfte durch die in dieser Arbeit verwendeten Access Pattern liegen. Um die beste Performance aus dem Globalspeicher zu erzielen, ist es nötig, bestimmte Zugriffsmuster auf den Speicher durchzuführen. Das ist jedoch nicht immer möglich (z.B. bei einer Distanzmatrix beim CVRP). Daran knüpft der Texturspeicher an, mit dem auch andere Zugriffsmuster, als die, die vom Globalspeicher gefordert sind, zügiger abgearbeitet werden können (Nvidia, 2012, S. 73). Er ist insbesondere für 2D-Zugriffsmuster ausgelegt und genau dieses Muster kommt beim Einsatz einer Distanzmatrix im Rahmen des CVRPs zum Tragen. Für den zügigen Datentransfer aus dem Globalspeicher ist es hingegen notwendig, dass die Daten, die gelesen werden, möglichst zusammenhängend gespeichert sind, was im Falle einer Distanzmatrix nicht zutreffen sollte.

2.7.4 Programmiermodell Compute Unified Device Architecture (CUDA)

Um die Ausführungen zur Implementierung nachvollziehen zu können, ist neben den grundlegenden Hardwarekenntnissen ein grober Überblick über das verwendete Programmiermodell nötig. Ein Programmiermodell kann nach McCool (2008, S. 817) als „abstract model of computation that is used by the programmer to reason about how a program executes“ definiert werden. Es ist also von fundamentaler Bedeutung, das zugrunde liegende Modell zu verinnerlichen, wenn ein möglichst performantes Programm geschrieben werden soll. Zur Programmierung einer Nvidia-GPU kommen im Wesentlichen OpenCL und CUDA in Betracht.[65] Beide Sprachen bzw. Spracherweiterungen für C sind nach Kim und Bond (2009, S. 9, Abb. 12) im Vergleich zu C mit nur wenig Mehraufwand bei der Einarbeitung verbunden, liefern dafür jedoch einen vergleichsweise großen Speedup. Der Vorteil von OpenCL ist die Plattformunabhängigkeit, die jedoch auf Nvidia-GPUs durch einen teilweise erheblichen Geschwindigkeitsverlust erkauft werden

[64] Selbst kostengünstige GPUs besitzen Speichergrößen von einem Gigabyte.

[65] Zu Beginn der GPU-Programmierung (für allgemeine Berechnungen) kamen Schnittstellen wie OpenGL oder DirectX zum Einsatz. Diese sind jedoch vornehmlich auf die Grafikausgabe ausgelegt, sodass die Nutzung dieser Schnittstellen sehr umständlich ist.

muss (vgl. Karimi et al. (2010) und Du et al. (2012)). Aufgrund dieser Tatsache und der Einschränkung durch die verwendete Hardware soll in dieser Arbeit CUDA zum Einsatz kommen, was im Folgenden erläutert wird.[66]

Um Berechnungen auf Grafikkarten auszuführen, ist ein sog. Kernel nötig. In diesem wird definiert, welche Aufgaben parallel erledigt werden sollen. Beim Aufruf des Kernels werden neben programmspezifischen Parametern zwei CUDA-spezifische Parameter, nämlich die Grid- und Block-Dimension, übergeben. Abbildung 2.28 stellt den schematischen Aufbau der beiden Dimensionen beispielhaft dar. Wie ersichtlich, setzt sich ein Grid aus einem oder mehreren Blöcken zusammen, die über einen

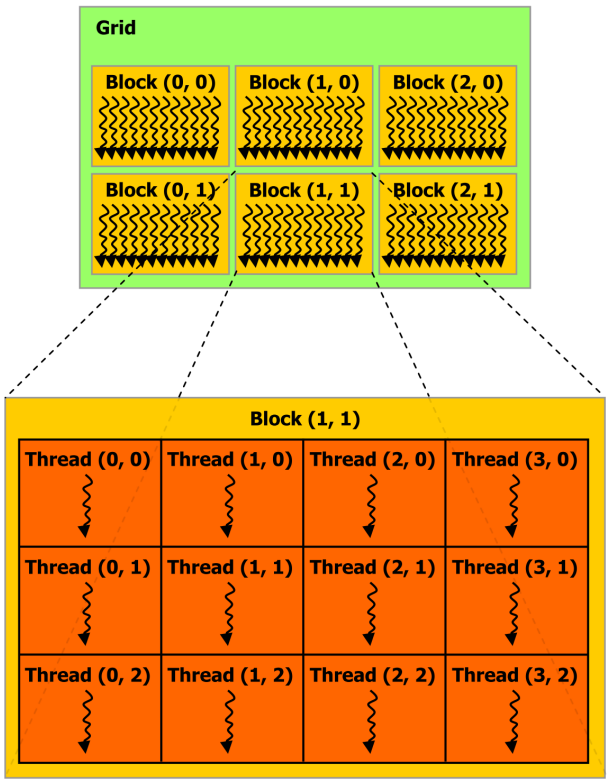


Abbildung 2.28: Programmiermodell CUDA (Quelle: Nvidia (2011b, S. 9))

dreidimensionalen Vektor identifiziert werden können. Ein Block wiederum setzt sich aus einem oder mehreren Threads zusammen, die ebenfalls mittels eines dreidimensionalen Vektors identifiziert werden. Die Threads werden von den SPs abgearbeitet, wobei keine Aussage darüber getroffen werden kann welcher Thread bzw. Threadteil von welchem SP ausgeführt wird. Wie in Abschnitt 2.7.3 beschrieben, besitzt jeder SP lokalen Speicher, der nur von ihm gelesen werden kann. Die Kommunikation und Synchronisation zwischen Threads ist nur eingeschränkt über das SMem möglich. Das heißt, mit Hilfe des SMem ist es für Threads innerhalb eines Blocks möglich, Daten auszutauschen, wohingegen der Datenaustausch über Blockgrenzen hinweg nicht ohne Weiteres möglich ist. Jeder Block wird genau einem SM zugeordnet, wobei sich diese Zuordnung im Laufe der Kernelabarbeitung nicht mehr verändert, d.h., ein Block wird einmalig einem SM zugeordnet und vollständig von ihm abgearbeitet.

An dieser Stelle soll das Scheduling nochmals näher betrachtet werden. Die Schedulingeinheit aus Abbildung 2.25 sorgt für die Verteilung der Blöcke auf die SMs bevor die Blöcke selbst das Scheduling

[66] Die Unterschiede zwischen OpenCL und CUDA sind nur sehr gering, sodass eine entsprechende Portierung mit geringem Zeitaufwand möglich sein sollte.

(mittels des Warp-Schedulers) ihrer Threads vornehmen. Dabei ist nicht vorhersehbar, wann welcher Block aktiv ist.

Es gibt noch einige erwähnenswerte Einschränkungen bzgl. der Dimensionen der Blöcke und Threads. So sind maximal 65535 Blöcke und maximal 1024 pro x- und y-Dimension der Threads möglich. Die z-Dimension des Threads ist auf 64 beschränkt. Gleichzeitig wird die Gesamtanzahl der Threads pro Block auf ebenfalls 1024 beschränkt, d.h., wenn die x-Dimension 1024 beträgt, kann die y- und z-Dimension jeweils lediglich 1 betragen. Weitere Details zu Beschränkungen der Hardware finden sich in Nvidia (2012, S. 136-139). Diese Beschränkungen sind sehr wichtig, wenn im weiteren Verlauf die Optimierung des GPU-Codes vorgenommen bzw. verstanden werden muss.

Einen guten Einstieg in die Grafikkartenprogrammierung mit CUDA bieten die Bücher von Sanders und Kandrot (2010) und Kirk und Hwu (2010).

3 Implementierung von Algorithmen zur Lösung von CVRPs

In diesem Kapitel wird auf die Umsetzung der Software in dieser Arbeit eingegangen. Es werden sowohl der Aufbau der CPU-Versionen als auch der Aufbau der GPU-Versionen im Detail vorgestellt. Ebenfalls werden die Unterschiede, die sich durch die Verwendung verschiedener Hardwarearchitekturen ergeben, aufgezeigt. Zunächst wird auf die Implementierung bzw. die Datenstrukturen der parallelen lokalen Suchoperatoren eingegangen. Darauf aufbauend werden Metaheuristiken beschrieben, die diese verwenden und dabei noch viele Aufgaben auf der CPU erledigen. Im letzten Schritt werden Algorithmen gezeigt, die möglichst wenig von der CPU Gebrauch machen und die Hauptarbeit bei der GPU liegt.

3.1 Lokale Suchoperatoren

Die Beschreibung der Umsetzung der lokalen Suchoperatoren in dieser Arbeit findet sich in Kapitel 2.3.2.2. Dabei wird nur auf die allgemeine Berechnung bzw. Ermittlung der Moves eingegangen und wie die Kapazitäten berechnet werden können. An dieser Stelle sollen zunächst die konkret verwendeten Datenstrukturen erläutert werden. Danach folgend wird der grundsätzliche Ablauf des Operators zur Bestimmung des besten Moves beschrieben, bevor dann die Besonderheiten, die im Rahmen der Implementierung einer GPU-Variante auftreten, beschrieben werden.

3.1.1 Datenstrukturen bei Anwendung lokaler Suchoperatoren

Das Verständnis über die Datenstrukturen ist von großer Bedeutung, um die Menge an Daten, die im späteren Verlauf von der CPU zur GPU transferiert werden, einschätzen zu können.

Zunächst einmal gilt es, das CVRP abzubilden. Dazu werden Arrays mit der Länge der Anzahl der Knoten gespeichert. Diese enthalten die Nachfrage d, die x-Koordinate und die y-Koordinate eines jeden Knotens. Neben der x- und y-Koordinate wird zusätzlich eine Distanzmatrix $dist$ der Größe $(n+1) \cdot (n+1)$ gespeichert, um möglichst schnell die Distanz zwischen zwei Knoten bestimmen zu können; die Distanzmatrix wird nur einmal beim Einlesen auf Basis der Koordinaten der Probleminstanz erzeugt und danach werden die Distanzen aus dem Speicher gelesen.[67] Bis auf die maximal zulässige Kapazität C der Fahrzeuge sind keine weiteren Datenstrukturen zur Problemrepräsentation notwendig.

Nachdem die Problemstellung eingelesen ist, muss eine Lösung repräsentiert werden, was mittels eines hinreichend großen Arrays erfolgt.[68] Hier wird von v_0 als Repräsentant des Depots abgesehen, sondern die Depots werden aufsteigend ab v_{n+1} durchnummeriert. In Abbildung 3.1 ist beispielhaft ein

[67] Das gilt zunächst für die Version, die nur die CPU verwendet. Die Verwendung einer vorberechneten Distanzmatrix auf der GPU wird erst im Folgenden evaluiert.

[68] Die Arraygröße ist dabei so dimensioniert, dass theoretisch jeder Kunde von einem Fahrzeug bedient wird, ohne dass es in diesem Fall zu einem Speicherüberlauf kommt.

6	5	1	7	2	8	3	4	8

Abbildung 3.1: Aufbau des Lösungsarrays

Datenstruktur	Relocate	Swap	OrOpt	2Opt*	2Opt	Cross-Exchange
$dist$ bzw. x, y	ja	ja	ja	ja	ja	ja
d	ja	ja	ja	nein	nein	ja
e_1	ja	ja	ja	ja	ja	ja
e_2	ja	ja	ja	ja	ja	ja
C	ja	ja	ja	ja	ja	ja
s	ja	ja	ja	ja	ja	ja
I	ja	ja	ja	ja	ja	ja
TID	ja	ja	ja	ja	ja	ja
C_r	ja	ja	ja	nein	nein	nein
$\overrightarrow{d}$	nein	nein	nein	ja	ja	ja
$\overleftarrow{d}$	nein	nein	nein	ja	ja	ja

Tabelle 3.1: Datenstrukturen der Operatoren

Lösungsarray s mit einem Problem von fünf Kunden dargestellt. Wie man sieht, ist die Lösung als große Tour kodiert. Das Depot beginnt mit 6 und wird mit jeder neuen Route um 1 erhöht. Der Grund, warum das Depot nicht als v_0 repräsentiert wird, liegt darin, dass die Moves mittels Generatorkanten bestimmt werden. Wenn nun eine Generatorkante von v_0 zu einem Knoten gespeichert ist, so ist nicht eindeutig festzustellen, von welchem Depot bzw. von welcher Route aus die Erzeugerkante ausgeht. Deshalb ist es nötig, die Routen einem „eindeutigen“ Depot zuzuordnen. Hier werden analog zu Irnich et al. (2006, S. 2412) und Alba und Dorronsoro (2004, S. 15) Kopien des Depots erzeugt, um die einzelnen Routen zu separieren. Die erzeugenden Kanten der Moves werden ebenfalls zu Beginn in zwei Arrays gespeichert, wobei ein Array e_1 den Startknoten der Kante und das andere Array e_2 den Endknoten der Kante enthält.

Grundsätzlich reicht die Repräsentation der Lösung wie oben beschrieben aus, um alle Informationen, die notwendig sind, direkt zu berechnen. Allerdings bietet es sich an, weitere Daten - insbesondere oft benötigte - im Speicher abzulegen. So wird die Kapazität C_r jeder einzelnen Route in einem Array, das der Länge der Routenzahl der aktuellen Lösung entspricht, gespeichert. Weiterhin wird für jeden Knoten vermerkt, auf welcher Position er sich in der aktuellen Lösung befindet, d.h., der Index I der Knoten auf der großen Tour wird gespeichert. Auch die Tour auf der sich ein Knoten befindet, wird in einem Array TID gespeichert, sodass durch einen einfachen Lookup die Route auf der sich ein Knoten befindet, bestimmt werden kann. Das ist insbesondere zur Bestimmung der Routenkapazität erforderlich, da diese Information nicht ohne eine RoutenId gefunden wird. Schließlich werden noch zwei Arrays gespeichert, die für jeden Knoten jeweils die Vorwärts- und Rückwärtskapazität $\overrightarrow{d}$ und $\overleftarrow{d}$ enthalten.

Tabelle 3.1 enthält eine Auflistung welche Daten von welchem Operator benötigt werden, um die Moves zu bewerten. Die ersten fünf Datenstrukturen $dist$, d, e_1, e_2 und C müssen dabei nicht bei einer Änderung der Lösung neuberechnet werden, wohingegen sich der Rest der Daten ändert, sobald sich die

Lösung verändert.[69] Die Neuberechnung findet im Vergleich zur Evaluation jedes einzelnen Moves i.d.R. so selten statt, sodass diese vernachlässigt werden kann.

3.1.2 Ablauf zur Bestimmung des besten Moves

Nachdem nun alle Informationen bzgl. der Operatoren und der notwendigen Datenstrukturen vorgestellt sind, wird nun darauf eingegangen, wie der Ablauf zur Bestimmung des besten Operators vonstatten geht.

In Algorithmus 5 ist der Sachverhalt visualisiert. In der Initialisierungsphase werden sämtliche Datenstrukturen generiert, die zur Bewertung der Moves benötigt werden. Danach wird jede erzeugende Kante auf Ihre Tauglichkeit als bester Move untersucht. Das Verfahren selektiert dazu eine Erzeugerkante und bestimmt, ob der daraus resultierende Move gültig ist. Sollte das der Fall sein, wird überprüft, ob die Distanz bzw. Kostenänderung im Vergleich zum bisher besten gefundenen Move günstiger ist. Ist die Distanz der so zu erzeugenden neuen Lösung geringer, wird die Generatorkante als neue beste Kante gesetzt. Wie beschrieben endet das Verfahren, wenn alle erzeugenden Kanten durchlaufen sind. Als Resultat liefert das Verfahren die beste Erzeugerkante oder $\emptyset$. Letzteres tritt dann ein, wenn alle erzeugenden Kanten zu keinem gültigen Move führen.

Algorithmus 5: Pseudocode zur Bestimmung des besten Moves

```
1  Initialisierung
2  Setze besten Move b = ∅
3  while Weitere Erzeugerkanten? do
4  |  Wähle nächste Erzeugerkante e
5  |  if e erzeugt gültigen Move then
6  |  |  if (Distanz von e < Distanz von b) oder (b = ∅) then
7  |  |  |  b = e
8  |  |  end
9  |  end
10 end
11 Ergebnis bester Move b
```

3.1.3 Lokale Suchoperatoren auf der Graphics Processing Unit (GPU)

Im vorangegangenen Abschnitt und Abschnitt 2.3.2.2 wurden die lokalen Suchoperatoren ausführlich beschrieben, weshalb an dieser Stelle nicht nochmals darauf eingegangen wird. Die grundlegenden Eigenschaften der GPU-Version unterscheiden sich nicht von der der CPU. An dieser Stelle soll der Fokus auf der Parallelisierung der Operatoren liegen. Dazu soll zunächst erläutert werden, wie die Operatoren, d.h. die Bewertung der Moves an sich parallelisiert wird, bevor darauf folgend erläutert wird, wie der beste Move mittels GPU bestimmt werden kann. In Abbildung 3.2 ist der Ablauf zur

[69] Natürlich kann der Fall eintreten, dass durch eine Veränderung der Lösung eine Kante aus e_1 bzw. e_2 nicht mehr erlaubt ist, dieses wird jedoch in den einzelnen Methoden, die die Moves bewerten, gesondert behandelt, da ein Update der Kantenliste sehr teuer - insbesondere im Hinblick auf die Arbeit mit der GPU - wäre.

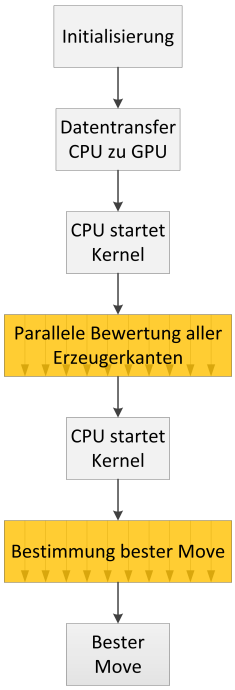


Abbildung 3.2: Flussdiagramm zur Parallelisierung der Bewertung und Bestimmung des besten Moves

Bestimmung des besten Moves dargestellt. Die Schritte, die auf der GPU durchgeführt werden, sind in orange dargestellt und werden im Folgenden näher betrachtet.

3.1.3.1 Parallelisierung lokaler Suchoperatoren

Die Parallelisierung ist in dieser Arbeit nicht operatorspezifisch, sondern lässt sich gleichermaßen auf alle vorgestellten Operatoren anwenden.

Als Ausgangspunkt dafür dient die Kantenliste, die die erzeugenden Kanten enthält. Bereits bei einer Probleminstanz von nur 100 Kunden müssen pro Iteration, in denen der Operator angewendet werden soll, über 10000 Kanten als potentielle Erzeugerkanten betrachtet werden. Es bietet sich also an, dass man die Kanten parallel bearbeitet bzw. bewertet, um somit möglichst viele Rechenkerne der GPU auszulasten.

Dazu werden so viele Threads gestartet, damit jeder genau eine Kante betrachten kann. Angenommen, es sollen 10000 Kanten bewertet werden und pro Block werden 512 Threads gestartet, so folgt daraus, dass $\frac{10000}{512} \approx 20$ Blöcke gestartet werden. Nur so ist in diesem Beispiel gewährleistet, dass alle Kanten durch jeweils einen Thread betrachtet werden.

Abbildung 3.3 zeigt die Ermittlung der Kante, die ein Thread bewerten muss. Wie bereits erwähnt, sind die Kanten in zwei Arrays, deren Länge der Anzahl der Erzeugerkanten entspricht, gespeichert. Damit kann man mittels der BlockId in x-Richtung sowie der ThreadId in x-Richtung für jeden Thread eine eindeutige KantenId ermitteln. Die KantenId ergibt sich aus der BlockId in x-Richtung multipliziert mit der Anzahl der Threads ß, die pro Block gestartet werden, zuzüglich der ThreadId in x-Richtung.

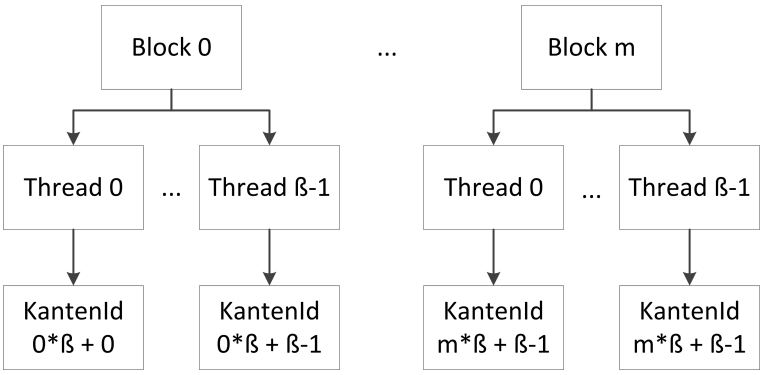


Abbildung 3.3: Ermittlung der zu bewertenden Kante eines Threads

Die y-Richtung hat in beiden Fällen, sowohl bei BlockId als auch ThreadId, eine Dimension von eins.[70]

Sobald die KantenId ermittelt ist, beginnt jeder Thread mit der Bewertung. Dabei wird auch berücksichtigt, ob die erzeugende Kante überhaupt in die aktuelle Lösung aufgenommen werden kann. Es ist bspw. möglich, dass die erzeugende Kante bereits in der Lösung enthalten ist. In diesem Fall wird die Kante als mögliche Erzeugerkante eines Moves verworfen. Das resultiert daraus, dass die Kantenliste mit den Generatorkanten nicht aktualisiert wird, was auf dem Aufwand, den die Aktualisierung mit sich bringen würde, beruht. Weiterhin wird auf der Grafikkarte bestimmt, ob eine Restriktion verletzt ist. Sollte das der Fall sein, so wird die Änderung des Zielfunktionswertes so groß gesetzt bzw. in das Ergebnisarray geschrieben, dass garantiert ist, dass diese Kante nicht die beste sein kann. Schließlich wird die Distanzänderung ermittelt, sofern keine Restriktion verletzt ist. Diese Distanzänderung wird in ein Ergebnisarray geschrieben. Das Ergebnis wird dabei an die gleiche Stelle wie die KantenId gespeichert. Die eigentliche Berechnung der Kapazitätsverletzung und Distanzveränderung erfolgt genauso wie auf der CPU.

Nun gilt es, die beste erzeugende Kante zu selektieren und als Ergebnis zu liefern.

3.1.3.2 Bestimmung des besten Moves auf der GPU

Zur Bestimmung des besten Moves wird das Ergebnisarray, das im vorangegangen Kernelaufruf angelegt und mit den Distanzveränderungen belegt ist, verwendet. Zur Ermittlung des Ergebnisarrays ist keine Synchronisierung zwischen den einzelnen Threads nötig, da sie vollkommen unabhängig voneinander ihre Werte bestimmen können. Hier kann das SIMD seine Stärke ausspielen. Etwas anders sieht es bei der Bestimmung des Maximums, d.h. der Bestimmung der besten Veränderung aus. In diesem Fall ist es unerlässlich, während des Ablaufs die Threads zu synchronisieren. Eine Fähigkeit, die den GPUs von Nvidia fehlt, ist die blockweite Synchronisation. Man kann keine Barriere aufbauen, an der jeder Thread wartet, bis alle anderen Threads, die noch nicht zu der Barriere gelangt sind, ebenfalls an die Barriere gelangen. Es besteht nur die Möglichkeit, Threads innerhalb eines Blocks zu synchronisieren. Um eine blockweite Synchronisierung zu erreichen, bietet sich die Möglichkeit an, zu

[70] Durch die Einschränkung der Hardware folgt, dass bei diesem Aufbau maximal $65535 \cdot 1024 = 67107840$ Threads gestartet werden können. Daraus folgt wiederum, dass maximal Instanzen bis ca. 8000 Kunden auf diese Art bearbeitet werden können. Da die größte Instanz in dieser Arbeit 1000 Kunden enthält, spielt diese Einschränkung hier keine Rolle. Um noch größere Instanzen als solche mit 8000 Knoten zu bearbeiten, ist somit ein neuer Entwurf des hier vorgestellten Algorithmus vonnöten.

warten, bis der Kernel auf der GPU beendet ist und dann direkt den nächsten Kernel zu starten. In diesem Fall wird die Synchronisierung über mehrere Kernelstarts erreicht. Dieses Vorgehen findet auch in der vorliegenden Arbeit Anwendung.

Um das Maximum[71] aus dem Ergebnisarray auf der GPU zu bestimmen, kommt ein Reduktionsalgorithmus zum Einsatz. Diese Art, das Optimum zu bestimmen, wird bereits in der Arbeit von Hillis und Steele (1986) eingesetzt. Die Funktionsweise des Verfahrens soll das Beispiel in Abbildung 3.4 verdeutlichen. Gegeben ist das Ergebnisarray, das die Bewertungen der einzelnen Moves enthält, die im vorangegangenen Schritt berechnet wurden. Nun wird auf der GPU ein Kernel gestartet, bei dem wieder mittels der x-Dimension des Blocks und Threads jedes Element erreicht wird. In dem dargestellten Beispiel bestimmt im ersten Durchgang jeder zweite Thread das Maximum von zwei Elementen; Thread 0 betrachtet in diesem Fall im Ergebnisarray die Elemente 0 und 1 und speichert das Ergebnis an Position 0 (aus Blocksicht betrachtet), Thread 1 befindet sich im Leerlauf und Thread 2 würde die Elemente 2 und 3 betrachten und an Position 2 schreiben usw.[72] Nachdem alle Threads das jeweilige Maximum im ersten Durchgang ermittelt haben, arbeitet nun die Hälfte der Threads weiter, die noch Zahlen enthält, bzw. die in der vorangegangenen Iteration mit Daten gefüllt wurden. Der Kernel endet, sobald nur noch zwei Elemente von Thread 0 in dem jeweiligen Block zu vergleichen sind und in das Array an der Position, die der BlockId entspricht, geschrieben sind. Nun hat jeder Block sein Maximum bestimmt. Um weiter fortzufahren wird der Kernel erneut aufgerufen. Hierbei muss nun eine geringere Anzahl an Blöcken gestartet werden, da die Daten bereits reduziert sind. Das Verfahren wiederholt sich so lange bis nur noch ein Block im Kernel aufgerufen werden muss, um alle übrigen Elemente zu betrachten. Das Maximum befindet sich dann an der 0-ten Position im Ergebnisarray.

Tabelle 3.2 zeigt einen möglichen Verlauf der Kernelaufrufe. Angenommen, es werden 10000 Kanten betrachtet und pro Block werden jeweils 64 Threads aufgerufen, dann müssen im ersten Durchlauf 157 Blöcke ausgeführt werden, damit alle Kanten Berücksichtigung finden. Diese 157 Blöcke liefern 157 Ergebnisse, die wieder als Ausgangsbasis für den nächsten Kernelaufruf dienen. Jetzt muss nur noch aus 157 Elementen das Maximum ermittelt werden und es sind nur noch 3 Blöcke notwendig, um alle Elemente zu erreichen. Damit verbleiben nach Beendigung des Kernels 3 Elemente, die ebenfalls noch überprüft werden müssen. Mit 64 Threads pro Block wird dann noch ein Block zur Bestimmung des Maximums benötigt und das Verfahren endet nach dem dritten Kernelaufruf.

Durchlauf	Anzahl Blöcke	Anzahl Threads pro Block	Anzahl Daten
1	$\frac{10000}{64} \approx 157$	64	10000
2	$\frac{157}{64} \approx 3$	64	157
3	$\frac{3}{64} \approx 1$	64	3

Tabelle 3.2: Blockkonfiguration zur Berechnung des besten Moves

Es wird deutlich, dass mit dem soeben beschriebenen Verfahren die Kapazitäten der GPU nicht vollständig ausgenutzt werden. Erkennbar ist das bspw. daran, dass Threads brachliegen und nicht an der Reduktion beteiligt sind. Um dieses zu verhindern, finden sich in dem Beitrag von Harris (o.J.) viele

[71] Auch wenn hier vom Maximum gesprochen wird, wird im Rahmen der Implementierung dieser Arbeit nach dem kleinsten c_Δ und somit dem Minimum gesucht.

[72] Das Vorgehen funktioniert auch blockübergreifend, da die ID im Ergebnisarray sich aus der BlockId und ThreadId in der x-Dimension ergibt.

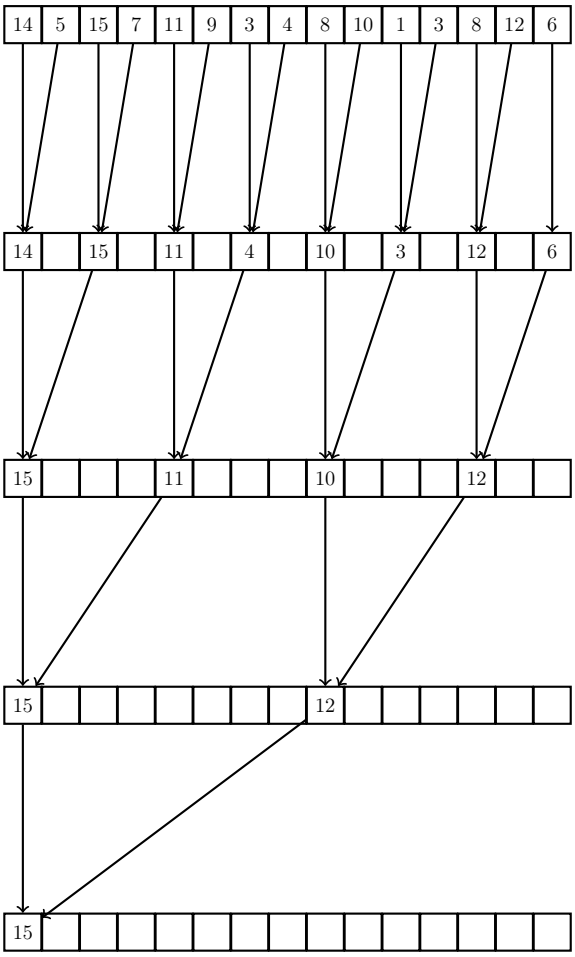


Abbildung 3.4: Reduktionsalgorithmus in Anlehnung an Harris (o.J.)

Optimierungen des hier vorgestellten Reduktionsalgorithmus für CUDA. Auch in der Implementierung wird der Code von Harris (o.J.) aus dem Nvidia GPU Computing SDK 4.2 weitgehend übernommen. Es soll jedoch an dieser Stelle nicht weiter auf Optimierungsmöglichkeiten eingegangen werden, da sie zum weiteren Verständnis der Arbeit nicht benötigt werden und nur ein grober Einblick in die Bestimmung des besten Moves gegeben werden sollte.

3.1.4 Spezifika der TS im Kontext lokaler Suchoperatoren

Bisher wird davon ausgegangen, dass in der Implementierung der Operatoren immer der beste zulässige Move selektiert wird. Das ist beim Einsatz der Verfahren, die auf $VNS \times SimA_{CPU}$ oder $VNS \times SimA_{GPU}$, die später beschrieben werden, basieren, erwünscht. Allerdings ergibt sich bei der TS eine Besonderheit. Wie in Abschnitt 2.3.3.2 beschrieben, wird im Falle der TS nicht der beste Move ausgehend von der aktuellen Lösung gesucht, sondern es wird eine zusätzliche Anforderung an den Operator gestellt. Er darf nicht tabu sein. An dieser Stelle soll die Implementierung der Tabuliste vorweggenommen werden, da diese direkt in die Operatoren integriert ist.

Angenommen, man bestimmt den besten Move ohne Berücksichtigung der Tabuliste im Rahmen einer TS, dann muss nach der Rückgabe des besten Moves geprüft werden, ob dieser tabu ist. In diesem Fall wird der Move verworfen und die TS läuft ohne Anwendung des Operators mit der aktuellen Lösung

weiter. Das kann dazu führen, dass danach wieder der gleiche Move selektiert und verworfen wird.[73] Dadurch befindet sich im Extremfall die TS in einer Endlosschleife. Um dem entgegenzuwirken, ist es wichtig, den Tabustatus eines Moves schon während der Bewertung desselben zu berücksichtigen. Das gewährleistet einerseits, dass nur Moves als Ergebnis geliefert werden, die auch erlaubt sind, und andererseits kann die Prüfung des Tabustatus weitere Prüfungen während der Bewertung des Moves ersparen, wenn der Tabustatus schon erkannt wurde. Das heißt, wenn ein Move tabu ist, muss nicht mehr die Einhaltung der Restriktionen überprüft werden, was gerade im Falle eines Cross-Exchange-Operators eine Zeitersparnis bringen kann. Allerdings muss hier auch beachtet werden, dass die so gewonnene Zeit durch die Prüfung des Tabustatus wieder reduziert wird.

Viele Verfahren in der Literatur setzen auf der Tabuliste bestimmte Merkmale, die eine Lösung innehaben kann, tabu (z.B. Cordeau und Maischberger (2012); Jin et al. (2012)). So kann z.B. verboten werden, dass ein Kunde, der gerade aus einer bestimmten Route entfernt wurde, wieder in diese Route eingefügt wird. In dieser Arbeit wird eine andere Form der Tabuliste, bei der ganze Lösungen tabu gesetzt werden, verwendet.

Um zu entscheiden, ob eine Lösung gleich einer anderen ist, werden die Zielfunktionswerte herangezogen. Es wird also nicht eine exakte Lösung tabu gesetzt, sondern vielmehr ihr Zielfunktionswert. Einerseits garantiert das, dass die Lösung tatsächlich tabu ist. Andererseits geht damit die Gefahr einher, dass eine Lösung, die gar nicht der tabu gesetzten Lösung entspricht, verworfen wird, da sie den gleichen bzw. ähnlichen Zielfunktionswert hat. Allerdings sorgt das auch dafür, dass mit dem Vorgehen eine breitere Diversifizierung erreicht werden kann. Zusätzlich dazu gibt es bei der Prüfung der Tabuliste noch zwei Parameter, die angeben in welchem Intervall alle Zielfunktionswerte tabu sind. Einen ähnlichen Ansatz verfolgt Prins (2004) im Rahmen seines evolutionären Algorithmus. Hierbei wird auch auf Grundlage des Zielfunktionswertes entschieden, ob sich zwei Lösungen unterscheiden.[74]

In Tabelle 3.3 sind beispielhaft einige Tabuwerte aufgeführt. Dabei wird angenommen, dass das Intervall, das tabu gesetzt wird, nach unten um 10 vom Zielfunktionswert abweicht und nach oben um 20. Daraus ergeben sich mittels des Zielfunktionswertes die Intervalle in denen Lösungen, deren Zielfunktionswert in dem Bereich liegt, tabu sind. Der Vorteil dieser Vorgehensweise liegt darin, dass man mittels der Parameter die Intervallgröße definieren bzw. im Suchverlauf ändern kann und somit die Diversifizierung dynamisch anpassen kann. Der Nachteil ist, wie bereits erwähnt, dass sehr viele Lösungen dadurch tabu sind und somit eventuell Lösungen nicht akzeptiert werden, über die zu sehr viel besseren Lösungen gelangt werden könnte. Dem Autor ist keine TS bekannt, die ein ähnliches Vorgehen wie das beschriebene vornimmt.

Nun stellt sich noch die Frage nach dem konkreten Ablauf bei der Prüfung der Tabuliste. Bekannt ist der aktuelle Zielfunktionswert der Lösung, die als Basis zur Bewertung des Moves dient. Die Distanz- bzw. Kostenveränderung wird, wie weiter oben beschrieben, in jedem Bewertungsschritt berechnet. Mit diesen beiden Informationen lässt sich der neue Zielfunktionswert, der durch Anwendung des Moves

[73] Dieser Fall dürfte vor allem dann eintreten, wenn lediglich ein Operator-Typ im Rahmen der TS verwendet wird.

[74] Wie in Abschnitt 2.3.3.2 beschrieben, ist die Überprüfung, ob zwei Lösungen exakt identisch sind, sehr aufwendig und speicherintensiv. Durch die Umsetzung in dieser Arbeit kann mit geringem Speicher- und auch Zeitaufwand bestimmt werden, ob sich zwei Lösungen ähnlich sind. Dabei wird jedoch in Kauf genommen, dass auch fälschlicherweise zwei Lösungen als identisch angenommen werden. Hier überwiegen jedoch die Performancevorteile, die sich durch die Abstellung auf den Zielfunktionswert als Entscheidungskriterium für den Tabustatus ergeben, weshalb sich für dieses Vorgehen entschieden wird.

Eintrag	Zielfunktionswert	Intervall
1	1000	[990; 1020]
2	950	[940; 970]
3	960	[950; 980]
4	920	[910; 940]
5	1050	[1040; 1070]

Tabelle 3.3: Beispielhafter Aufbau einer Tabuliste

resultiert, bestimmen. Mittels dieses Wertes lässt sich dann überprüfen, ob der Zielfunktionswert tabu ist. In diesem Fall wird der Move nicht weiter beachtet. Der Aufwand zur Prüfung der Tabuliste beträgt $O(TLL)$ mit TLL als Länge der Tabuliste.

3.2 Kombination einer VNS mit SimA

An dieser Stelle soll ein erstes Anwendungsgebiet der Operatoren im Rahmen einer Metaheuristik erläutert werden. Es wird vorwiegend aus der Perspektive der GPU-Version beschrieben. In der Arbeit wird differenziert zwischen einer VNS mit SimA $VNS \times SimA_{CPU}$ und $VNS \times SimA_{GPU}$. Die Benennung soll dabei nicht implizieren, dass die CPU-Variante alleinig von der CPU durchgeführt wird. Das „CPU“ im Namen drückt lediglich aus, dass die CPU in dieser Version stärker beteiligt ist, wohingegen die GPU-Variante fast allein von der GPU ausgeführt wird. Zunächst wird auf die $VNS \times SimA_{CPU}$ eingegangen, bevor die Details der $VNS \times SimA_{GPU}$ erläutert werden.

3.2.1 CPU-Variante: $VNS \times SimA_{CPU}$

Die einfachste Nutzung der lokalen Suchoperatoren besteht darin, jeweils den besten Move auf die aktuelle Lösung anzuwenden. Genau vor diesem Hintergrund wird als Basis die VNS gewählt. Wie bereits beschrieben, sind sechs verschiedene Operatoren vorhanden. Diese eignen sich hervorragend zum Einsatz in einer VNS, da sich die Moves soweit unterscheiden, dass man damit sehr unterschiedliche Nachbarschaftsstrukturen erzeugt - genauso wie es im Sinne der VNS gewünscht ist.

In Algorithmus 6 ist der Pseudocode der implementierten Metaheuristik dargestellt. Im Vergleich zur Basisversion der VNS finden sich einige Unterschiede. Der wohl bedeutendste besteht in der Alternierung der Nachbarschaften. Im Grundalgorithmus wird nur dann eine größere bzw. andere Nachbarschaftsstruktur gewählt, wenn keine bessere Lösung durch den besten Move mehr gefunden werden konnte. In der vorliegenden Implementierung wird hingegen die Nachbarschaft auch bei einem erfolgreichen Move verändert, wenn eine neue beste Lösung gefunden wird. Ein weiterer entscheidender Unterschied zur Standard-VNS ergibt sich daraus, wie die unterschiedlichen Nachbarschaftsstrukturen Anwendung finden. Im Grundalgorithmus werden die Nachbarschaften vornehmlich in der Shaking-Phase verwendet, wohingegen sie hier sozusagen in der Lokalsuche genutzt werden. Die Analogie zur VNS ergibt sich bei dem vorgestellten Algorithmus grundsätzlich durch den Gebrauch einer Lokalsuche und eines Shakings. Trotzdem findet sich eine weitere Parallele in der Alternierung der Nachbarschaftsstrukturen, mit dem Unterschied, dass sie in dieser Arbeit vornehmlich in der Lokalsuche zum Einsatz kommt. Nachdem alle Nachbarschaftsstrukturen einmal ihren jeweils besten Move als Ergebnis geliefert haben,

Algorithmus 6: Pseudocode der Basisversion der $VNS \times SimA_{CPU}$

```
1  Wähle Nachbarschaftsstruktur N_M, M = 1, ..., M_max
2  Setze Initiallösung s ∈ S
3  Wähle Abbruchbedingung
4  Setze Temperatur T
5  s' = s
6  while Abbruchbedingung nicht erfüllt do
7      k = 1
8      //hier beginnt die Lokalsuche
9      while M ≤ M_max do
10         s' durch besten Move der Nachbarschaft N_M(s')
11         if f(s') < f(s) then
12             s = s'
13         end
14         M = M + 1
15     end
16     //hier endet die Lokalsuche und es beginnt Shaking
17     Generiere zufällig i_l, j_l
18     Bewerte alle Moves der Nachbarschaft N_{Cross-Exchange_{i_l,j_l}}(s')
19     while Move angewendet oder Iterationszahl überschritten do
20         Bestimme zufällige Lösung s'' mittels der Menge der bewerteten Moves
21         if f(s'') < f(s') then
22             s' = s''
23         else
24             if rand(0,1) < e^{-(f(s')-f(s''))/T} then
25                 s' = s''
26             end
27         end
28     end
29     //hier endet Shaking
30 end
```

wird ein Shaking-Verfahren auf der aktuellen Lösung durchgeführt.

Die Initiallösung wird nach dem Verfahren von Clarke und Wright (1964) erzeugt, wobei hier auf das Framework, das von Groër et al. (2010) stammt, zurückgegriffen wird. Dieses ermöglicht es durch Variieren weniger Parameter, unterschiedliche Lösungen zu generieren, was vor allem für die im weiteren Verlauf der Arbeit vorgestellten Verfahren von großer Bedeutung ist.

Das Shaking-Verfahren verwendet zur Generierung einer neuen Lösung lediglich den Cross-Exchange-Operator. Dieser wird jedoch in verschiedenen Konfigurationen gestartet, d.h., beim Shaking kommt zufällig eine Kombination von i_l zwischen 1 und 2 und j_l zwischen 0 und 2 zum Einsatz.[75] [76] Nachdem die Konfiguration des Cross-Exchange-Operators bestimmt ist, werden auf der GPU alle Moves bewertet und zurück zur CPU transferiert. Aus diesen Moves wird einer zufällig ausgewählt. Sollte der Move die aktuelle Lösung verbessern, wird er sofort angenommen und ausgeführt. Wenn sich die Lösung verschlechtert, wird mittels einer Akzeptanzwahrscheinlichkeit, die auch im SimA zum Einsatz kommt, entschieden, ob er trotzdem durchgeführt wird. Sollte ein Move angenommen worden sein, wird mit der Lokalsuche fortgefahren bzw. der Algorithmus endet, falls die Abbruchbedingung erreicht ist. Wenn der Operator nicht ausgeführt wird, werden bis zu einer bestimmten Anzahl von Iterationen weiter zufällige Moves aus der bewerteten Liste selektiert und geprüft, ob sie als Shaking-Move dienen können. Erst wenn diese Anzahl überschritten ist, wird ohne Shaking fortgefahren.

Die Reihenfolge der Nachbarschaften N_M ist zufällig festgelegt, jedoch wird sie für die gesamten numerischen Studien konstant gehalten. Insgesamt kommen bei der Implementierung 30 bzw. 29 verschiedene Nachbarschaften zum Einsatz. Dazu zählen neben dem Relocate-, Swap-, Or-Opt-, 2-Opt- und 2-Opt*-Operator auch der Cross-Exchange-Operator. Letzterer sorgt durch seine unterschiedlichen Konfigurationsmöglichkeiten, d.h. die Länge der Kundensequenzen, die ausgetauscht werden, für eine erhebliche Steigerung der Anzahl der Nachbarschaftsstrukturen. Dabei haben Tests gezeigt, dass Kundensequenzen, die mehr als fünf Kunden enthalten, nicht zu befriedigenden Ergebnissen führen. Das lässt sich eventuell damit begründen, dass durch zu große Sequenzen die Lösungsstruktur sehr stark zerstört wird und es zunehmend weniger Optionen zur Durchführung des Operators gibt.[77] Aufgrund dessen wird die maximale Länge der Kundensequenz auf fünf und die minimale Länge auf eins beschränkt, wodurch 25 verschiedene Cross-Exchange-Operatoren resultieren.[78]

Da die bereits vorgestellten Komponenten allein noch nicht zu befriedigenden Ergebnissen führen, wird an dem $VNS \times SimA_{CPU}$ eine weitere Anpassung vorgenommen, die dem SimA entlehnt ist. Neben der Akzeptanzwahrscheinlichkeit, die bereits beim Shaking zum Einsatz kommt, nimmt eine Akzeptanzwahrscheinlichkeit auch Einfluss auf die Annahme der anderen Moves, die nicht im Kontext des Shakings durchgeführt werden. Das ist vor dem Hintergrund zu betrachten, dass im Laufe des Suchprozesses oftmals nur sehr viel schlechtere Lösungen als die aktuelle generiert werden. Damit man durch einen Move, den man auf jeden Fall annehmen muss, sich nicht zu weit von der aktuellen Lösung

[75] Größere Werte für i_l und j_l haben dazu geführt, dass die Lösung sich allzu sehr verschlechtert hat und es für das Suchverfahren nicht mehr möglich ist, bessere Lösungsbereiche zu finden.

[76] Es kommen also der 1,2-1,2,3-Cross-Exchange zum Einsatz.

[77] Durch das Kriterium, dass sich die Kundensequenz innerhalb einer Route befinden muss, ist eine Vielzahl von Generatorkanten nicht in der Lage einen gültigen Move zu erzeugen und somit verbleiben weniger Kanten, die zu einem überhaupt gültigen Move führen.

[78] Hier sollte noch angemerkt werden, dass der Swap-Operator durch den 1-1-Cross-Exchange ebenfalls abgedeckt ist. Der Einfachheit halber wird er somit zweimal beim Alternieren der Nachbarschaftsstrukturen durchlaufen.

entfernt, erscheint es sinnvoll, die Akzeptanz von Operatoren einzuschränken.

Algorithmus 7: Pseudocode der $VNS \times SimA_{CPU}$

```
1  Wähle Nachbarschaftsstruktur N_M, M = 1, ..., M_max
2  Setze Initiallösung s ∈ S
3  Wähle Abbruchbedingung
4  Wähle Initialtemperatur T
5  s' = s
6  while Abbruchbedingung nicht erfüllt do
7      M = 1
8      while M ≤ M_max do
9          bestimme besten Move aus Nachbarschaft N_M(s')
10         if bester Move verbessert aktuelle Lösung s' then
11             s' = wende besten Move an
12         else
13             if rand(0,1) < e^(-c_Δ / T) then
14                 s' = wende besten Move an
15             end
16         end
17         if f(s') < f(s) then
18             s = s'
19         end
20         M = M + 1
21         Aktualisiere T
22     end
23     Shaking
24 end
```

In Algorithmus 7 ist der angepasste Pseudocode dargestellt. Wie ersichtlich wird wie bisher von jeder Nachbarschaftsstruktur der beste Move bestimmt. Sobald er die aktuelle Lösung verbessert, wird der Move direkt auf die Lösung angewendet. Sollte der Operator jedoch die aktuelle Lösung verschlechtern, so wird er nur mit einer bestimmten Wahrscheinlichkeit ausgeführt. Die Wahrscheinlichkeit ergibt sich in Analogie zum SimA.

Durch die Einführung der Akzeptanzwahrscheinlichkeit ist es nötig, einen Abkühlungsplan einzuführen und zu bestimmen, wann welche Temperatur verwendet wird. In dem hier beschriebenen Verfahren wird die Temperatur nach einer festen Anzahl von Schritten mittels des exponentiellen Abkühlungsplans, wie in Formel 2.33 definiert, reduziert.

Abbildung 3.5 zeigt den Algorithmus nochmals in einem Flussdiagramm. Es wird ersichtlich, dass nur ein vermeintlich kleiner Teil des Gesamtverfahrens auf der GPU gerechnet wird (orange dargestellt). Allerdings stellt, wie bereits erläutert, dieser Teil den größten Rechenaufwand dar. Jeder einzelne Thread kümmert sich zunächst um die Bewertung der erzeugenden Kanten wie es in Abbildung 3.6 dargestellt ist. Die Kante, die von dem jeweiligen Thread bewertet werden muss, ergibt sich, wie schon

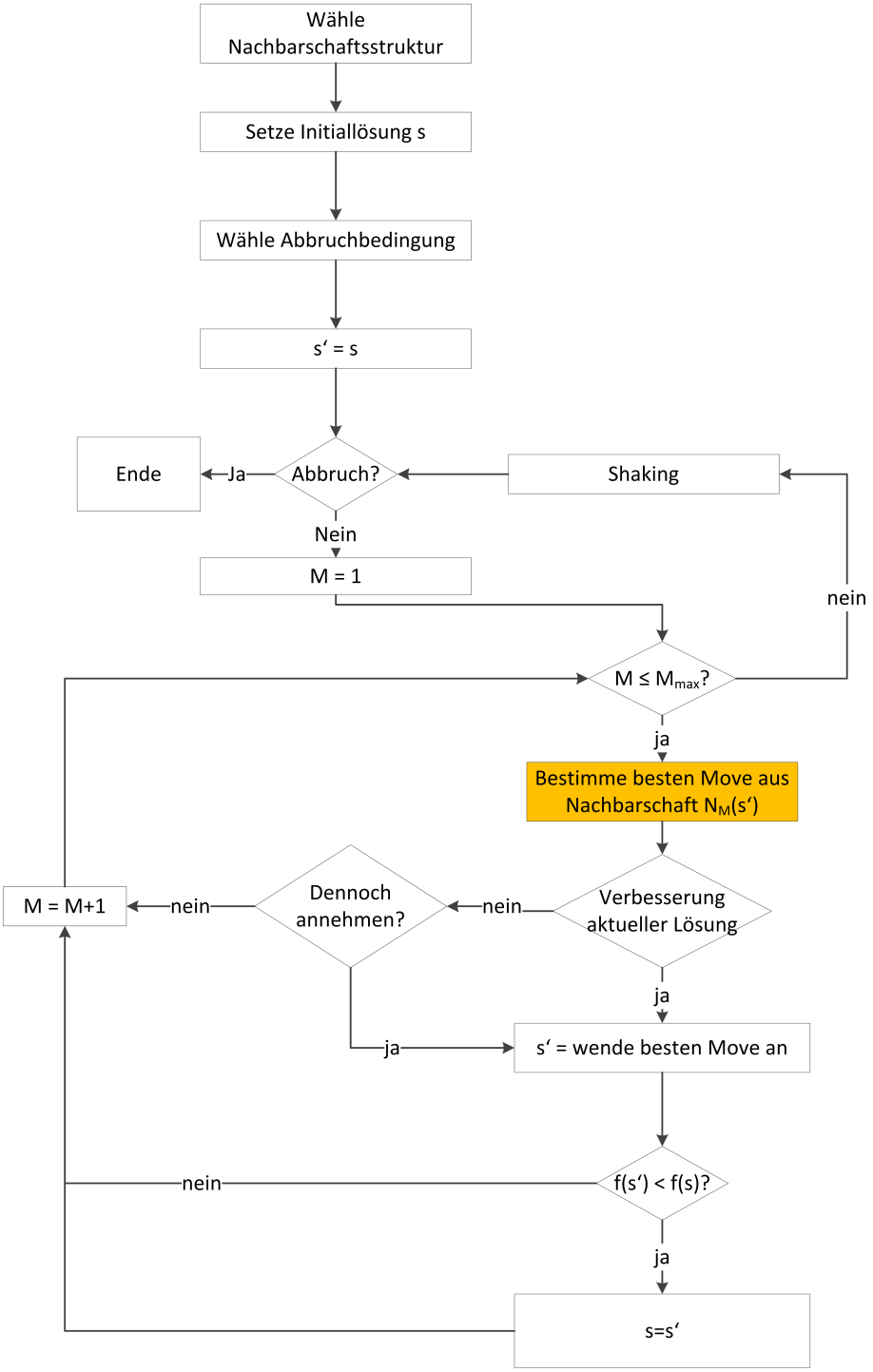


Abbildung 3.5: Flussdiagramm der $VNS \times SimA_{CPU}$

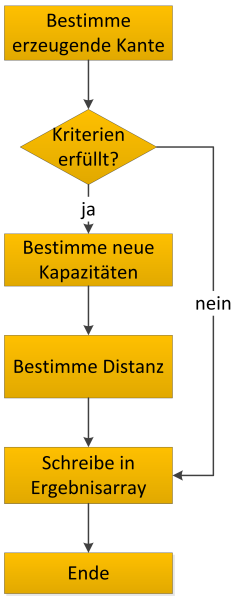


Abbildung 3.6: Bewertung der erzeugenden Kante in der $VNS \times SimA_{CPU}$

beschrieben, aus der BlockId, der Blockgröße und der ThreadId. Sobald die Kante festgelegt ist, müssen bestimmte Kriterien überprüft werden. Darunter fällt bspw. die Prüfung, ob die erzeugende Kante bereits in der aktuellen Lösung vorhanden ist und somit eine negative Bewertung festgestellt wird, da sie zu keiner neuen Lösung führen würde. Das heißt, wenn ein Kriterium nicht erfüllt ist, so wird im Ergebnisarray die Kante als nicht wählbar gespeichert. Sind jedoch alle Kriterien erfüllt, werden die neuen Kapazitäten, die durch den Move entstehen, sowie die Distanzveränderung ermittelt. Diese Daten bilden dann die Grundlage für das Ergebnisarray, d.h., wenn eine Kapazität überschritten ist, wird das im Ergebnisarray vermerkt.[79] Sollte keine Überschreitung vorliegen, wird die Distanz in das Array geschrieben und später als Entscheidungskriterium für die Bestimmung des besten Moves herangezogen. Die Bestimmung des Maximums findet, wie aufgeführt, ebenfalls auf der GPU statt und wird auf Grundlage des Ergebnisarrays ermittelt (siehe Abschnitt 3.1.3.2).

Die Verknüpfung der VNS mit SimA findet in Analogie zu Schneider (2012, S. 102) statt. Das Verfahren wird für das Electric-Vehicle-Routing-Problem with Time Windows vorgeschlagen. Der Autor prüft dabei mittels einer Akzeptanzwahrscheinlichkeit, ob die Lösung als neue Lösung angenommen wird. Zuvor findet ein Shaking-Move nach dem traditionellen Gedanken der VNS statt. Darauf folgt eine Lokalsuche, die mittels einer TS umgesetzt ist, bevor dann die Akzeptanz der Lösung überprüft wird.

[79] Hier wird in Analogie zu einer Strafkostenfunktion, die oft bei Verfahren, die ungültige Lösungen akzeptieren, eingesetzt wird, das Ergebnis gespeichert. Das Ergebnis in dem Array ergibt sich aus der Summe c_Δ und $L \cdot C_v$, wobei C_v die Kapazitätsverletzung, die durch den Move entsteht, angibt und L eine hinreichend große Zahl ist, die sicherstellt, dass ein ungültiger Move niemals als bester Move selektiert werden kann. Hier ist es also möglich, die vorgestellten Algorithmen in der Arbeit durch wenige Änderungen in Verfahren, die Strafkosten verwenden, zu nutzen.

3.2.2 GPU-Variante: $VNS \times SimA_{GPU}$

Auch wenn das vorangegangene Verfahren in Tests bereits sehr gute Ergebnisse liefert, (vgl. Abschnitt 4.2.1.1) hat sich gezeigt, dass durch diesen Aufbau bzw. durch diese Methode der Parallelisierung, die Hardwareressourcen der Grafikkarte nicht vollständig ausgenutzt werden. Es wird deutlich, dass die GPU - selbst während der Bewertung und Bestimmung des besten Moves - nur teilweise ausgelastet ist. Das veranlasste die Entwicklung einer Metaheuristik, die die Rechnerressourcen sehr viel besser ausnutzt.

3.2.2.1 Aufbau der $VNS \times SimA_{GPU}$

Die Grundidee des Verfahrens entspricht der bereits vorgestellten $VNS \times SimA_{CPU}$. Allerdings werden in dieser Version bedeutend mehr Aufgaben von der GPU übernommen. In der Variante $VNS \times SimA_{CPU}$ werden lediglich die erzeugenden Kanten bewertet und der beste Move ermittelt.

Die Bewertung der Gesamttour, d.h. die Bestimmung der Länge bzw. des Zielfunktionswertes, der Kapazitäten der Routen, der Tourindices usw. werden im vorangegangenen Verfahren vollständig auf der CPU ausgeführt und von dort auf die GPU transferiert. Das hier betrachtete Verfahren $VNS \times SimA_{GPU}$ besitzt die Fähigkeit, die Tourbewertung vollkommen autark auf der Grafikkarte durchzuführen, was den Datentransfer von der CPU zur GPU verringert und somit die Ausführungszeit reduzieren kann.

Weiterhin ist es der GPU-Variante möglich, im Vergleich zur $VNS \times SimA_{CPU}$ die Moves bzw. den besten Move, der ausgewählt wird, vollständig auf der GPU auszuführen, ohne dass eine weitere Interaktion der CPU vonnöten ist. Dadurch kann das Verfahren vollkommen selbstständig Iterationen der adaptierten $VNS \times SimA_{CPU}$ ausführen.

Um diese Selbstständigkeit der Grafikkarte zu erreichen, ist es notwendig, dass an bestimmten Stellen Threads aufeinander warten und Synchronisationsbarrieren gesetzt werden. Wie bereits beschrieben, ist das bei Nvidia-GPUs nur blockweise möglich. Das führt dazu, dass in dem so realisierten Aufbau der $VNS \times SimA_{CPU}$ lediglich ein Block gestartet werden kann, um alle Berechnungen auf der GPU auszuführen.[80] Jedoch bringt das wiederum den Nachteil mit sich, dass dieser eine Block genau einem SM zugeordnet wird. Bei Ausführung des Codes kommt es dazu, dass nur ein SM voll ausgelastet ist, und alle anderen verfügbaren im Leerlauf sind. Daher ist der Gedanke naheliegend, die anderen Prozessoren der Grafikkarte ebenfalls Lösungen bearbeiten zu lassen, weshalb eine Multistart-Lösung implementiert wird. Damit ist es möglich, dass jeder Block unabhängig von den anderen eine Lösung bearbeitet. Eine Synchronisation ist nur innerhalb eines Blocks notwendig, wodurch die Beschränkungen der Hardware abgebildet werden können.

Abbildung 3.7 zeigt den Aufbau des Multistart-Verfahrens. Man erkennt, dass bedeutend mehr Aufgaben auf die GPU ausgelagert werden. Lediglich die ersten Initiallösungen werden auf der CPU generiert. Dabei werden die Parameter, die das Framework von Groër et al. (2010) zur Generierung von Initiallösungen benötigt, zufällig bestimmt und somit sichergestellt, dass unterschiedliche Lösungen erzeugt werden. Danach werden diese neben den Problemdaten zur GPU transferiert und die GPU arbeitet autark an der Lösung des Problems. Trotzdem bietet es sich auch beim Multistart-Verfahren

[80] Bei den Ausführungen wird von der Beschränkung auf 1024 Threads pro Block abstrahiert. Wird diese zusätzlich betrachtet, sind bei Berechnung mittels eines Blocks maximal 1024 Kanten zu erreichen.

an, dass Lösungen - insbesondere beste Lösungen - ausgetauscht werden. Das geschieht nach einer zuvor definierten Anzahl von Iterationen, nach denen alle[81] Lösungen der einzelnen Blöcke eingesammelt und zur CPU übertragen werden. Diese entscheidet dann, welche Initiallösung welchem Block für den nächsten Durchlauf der VNS zugeordnet wird.

Abbildung 3.7: Allgemeiner Ablauf der $VNS \times SimA_{GPU}$

In Darstellung 3.8 ist der grobe Ablauf des Algorithmus auf Blockbasis, d.h. welche Aufgaben die Blöcke zu erledigen haben, dargestellt. Es muss unterschieden werden, ob ein Thread im Block die Arbeit erledigt (1 Pfeil), oder ob alle Threads eines Blocks (mehrere Pfeile) an einer Methode beteiligt sind. Die Initiallösung, die von der CPU stammt, wird von allen Threads eines Blocks in den lokalen Speicher (SMem) kopiert und als aktuelle Lösung gesetzt. Danach findet die Bewertung der Gesamttour statt, wobei lediglich ein Thread pro Block daran beteiligt ist. Dieser berechnet und stellt alle Informationen für alle anderen Threads im Block für die weitere Berechnung bereit. Diese findet im Vergleich zur Bewertung der erzeugenden Moves relativ selten statt, weshalb hier auf eine weitere Parallelisierung verzichtet wird und nur ein Thread daran beteiligt ist. Nachdem die aktuelle Lösung ausgewertet ist, berechnen alle Threads die Änderung des Zielfunktionswertes, der sich durch Zugrundelegung der aktuellen Tour und der Generatorkanten ergibt. Dabei übernimmt ein Thread im Gegensatz zum Verfahren $VNS \times SimA_{CPU}$ nicht mehr nur die Bewertung von genau einer Kante,

[81] Es werden nicht sämtliche Lösungen, die ein Block jemals gefunden hat, transferiert, sondern die bis dato beste gefundene Lösung des Blocks.

sondern von $\frac{\text{Anzahl der Kanten}}{\text{Anzahl der Threads pro Block}}$-vielen. Sobald diese Auswertungen abgeschlossen sind, wird der beste Move mittels aller Threads bestimmt. Nun wird in Analogie zu $VNS \times SimA_{CPU}$ geprüft,

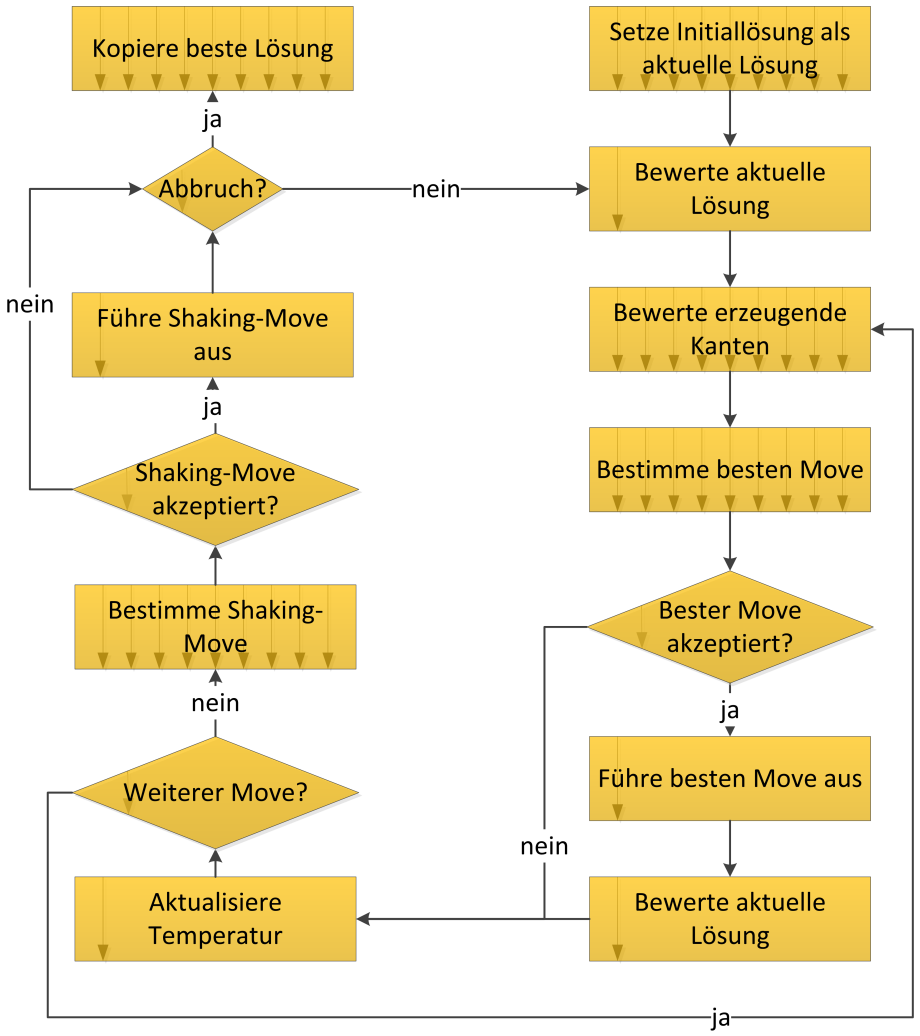


Abbildung 3.8: Ablauf des $VNS \times SimA_{GPU}$ auf Blockbasis

ob der Move akzeptiert wird, d.h., ob er die aktuelle Lösung verbessert oder wenn nicht, ob er trotzdem durch die Annahmewahrscheinlichkeit angenommen wird.Wenn der Move akzeptiert wird, wird er von einem Thread in dem Block ausgeführt. Die Temperatur wird mittels eines Threads nach Absuchen einer Nachbarschaftsstruktur angepasst. Nachdem alle Nachbarschaftsstrukturen einmal abgesucht sind, wird der Shaking-Move von allen Threads in Analogie zur $VNS \times SimA_{CPU}$ ermittelt. Sollte er akzeptiert werden, wird er ebenfalls von einem Thread ausgeführt.[82] Wenn die Abbruchbedingung nicht erfüllt ist, wird auf der GPU mit der nächsten Iteration fortgefahren. Andernfalls wird die beste gefundene Lösung während des Kernelaufrufs innerhalb des Blocks von allen Threads des Blocks in den globalen Speicher kopiert, wodurch dann wieder ein Zugriff durch die CPU realisiert werden kann.

Auf der CPU wird dann entschieden, was mit den Lösungen, die auf der GPU generiert wurden, geschieht. Das bedeutet es muss geprüft werden, welche Lösungen in den Lösungspool aufgenommen

[82] Auch bei der Ausführung der Moves bringt die Parallelisierung sehr viele Probleme mit sich, die nicht im Verhältnis zur Zeitersparnis stehen, da die Move-Anwendung ebenfalls nur relativ selten zur Anwendung kommt. Deshalb wird an dieser Stelle auf eine weitere Parallelisierung verzichtet.

werden und welche alten Lösungen ersetzt werden. Weiterhin muss die CPU entscheiden, welche Lösungen für den nächsten Durchgang auf der GPU - sofern noch weitere durchzuführen sind - verwendet werden. Dabei wird in dieser Arbeit grob differenziert zwischen der Möglichkeit, dass alle Blöcke die bisher beste gefundene Lösung als Initiallösung verwenden oder, dass sie mit ihrer aktuellen Lösung weiterarbeiten. Die Verteilung der besten Lösung als Initiallösung sorgt, der stochastischen Natur der Annahmewahrscheinlichkeit geschuldet, keineswegs dafür, dass alle Blöcke mit der gleichen Lösung enden. Die Lösungswege sind trotz gleicher Initiallösung sehr unterschiedlich, wodurch auch dann eine Exploration des Lösungsraumes erreicht wird.

Ein Großteil der benötigten Datenstrukturen wird im SMem abgelegt. Durch den zumeist zufälligen Zugriff auf diese Daten kann im Vergleich zum Globalspeicher kein bzw. nur ein sehr geringer Geschwindigkeitsgewinn erzielt werden. Es erleichtert jedoch erheblich die Verwaltung bzw. die Kapselung der Daten zwischen den Blöcken.

3.2.2.2 Implementierung eines adaptiven Abkühlungsplans in der $VNS \times SimA_{GPU}$

Durch die Verwendung der Metropolis-Wahrscheinlichkeit ist es notwendig, die Temperatur im Verlauf des Suchprozesses anzupassen. Hierbei sind zwei unterschiedliche Methoden in der Variante $VNS \times SimA_{GPU}$ implementiert. Zum einen wird die Temperatur in Analogie zu $VNS \times SimA_{CPU}$ nach einer bestimmten Anzahl von Iterationsschritten bzw. Auswertung von Nachbarschaften mittels Formel 2.33 angepasst. Hierbei ist also bereits zu Beginn festgelegt, wann welches Temperaturniveau erreicht wird.

Auf der anderen Seite ist es oftmals wünschenswert, dass sich die Temperatur adaptiv, d.h. in Abhängigkeit von den generierten Lösungen, selbst anpasst. Dazu wird in dieser Arbeit der gleitende Durchschnitt nach folgender Formel verwendet (Wendt, 1995, S. 151):

$$\overline{c_\Delta}(g) = \frac{1}{m} \sum_{i=g-m+1}^{k} c_\Delta(i) \tag{3.1}$$

Je nach Wert von $\overline{c_\Delta}(g)$ ergibt sich folgende Interpretation[83]: „Weist der gleitende Durchschnitt einen negativen Wert auf, so wird davon ausgegangen, dass sich das System noch auf dem Weg zum thermodynamischen Gleichgewicht befindet, das Temperaturniveau T wird konstant gehalten. Ist die Energie dagegen während der letzten m Transitionsversuche gleich geblieben oder angestiegen, ist also der gleitende Durchschnitt positiv, so gilt das thermodynamische Gleichgewicht als erreicht, die Temperatur kann folglich abgesenkt werden“ (Wendt, 1995, S. 151). Die Abkühlung als solche findet weiterhin in Analogie zu Formel 2.33 statt. Der hier implementierte adaptive Abkühlungsplan sorgt lediglich für eine Anpassung der Temperatur nach einer variablen Anzahl von Iterationen. Für die Nutzung eines solchen Abkühlungsplans scheint im ersten Moment zu sprechen, dass man nun nicht mehr festlegen muss, wann die Temperatur gesenkt wird, da dies automatisch geschieht. Trotzdem kommt es nicht zu einer Parameterreduzierung, da im Rahmen des dynamischen Plans bestimmt werden muss, wie viele Iterationsschritte zur Berechnung des gleitenden Durchschnitts berücksichtigt werden sollen.

Der adaptive Abkühlungsplan wird weniger aufgrund der Aussicht auf eine höhere Lösungsgüte in dieser Arbeit implementiert, sondern vielmehr um eine einfache Erweiterbarkeit der implementierten

[83] $c_\Delta(i)$ gibt die Veränderung des Zielfunktionswertes in der i-ten Iteration an.

Verfahren zu ermöglichen sowie um die Auswirkungen, die ein adaptiver Abkühlungsplan auf die Performance hat, zu messen. Durch den zusätzlichen Aufwand, der zur Berechnung des gleitenden Durchschnitts erforderlich ist, kommt es zu einer Reduzierung der Performance, die aus der Perspektive der GPU quantifiziert werden soll.[84]

Des Weiteren speichert jeder Block seine eigene Temperatur - unabhängig davon, ob ein statischer oder dynamischer Abkühlungsplan eingesetzt wird. Das ermöglicht eine breite Differenzierung des Suchprozesses zwischen den Blöcken. Auch wird die Temperatur zu Kernelbeginn aus dem globalen Speicher gelesen bzw. zu Kernelende geschrieben, damit der Algorithmus bei mehreren aufeinanderfolgenden Kernelaufrufen die zuletzt verwendete Temperatur nicht „vergisst“.

3.3 Umsetzung einer TS

An dieser Stelle soll die TS, die im Rahmen dieser Arbeit implementiert wird, erläutert werden. Auch hier wird in Analogie zur Beschreibung der VNS der Fokus auf der GPU-Seite liegen. Die TS_{CPU} hat ebenso einen GPU-Anteil inne, der jedoch im Vergleich zu dem darauf folgenden beschriebenen Verfahren TS_{GPU} einen sehr viel geringeren Berechnungsteil auf die GPU auslagert.

3.3.1 CPU-Variante: TS_{CPU}

Die TS TS_{CPU} ist sehr ähnlich zu $VNS \times SimA_{CPU}$ aufgebaut; es werden sowohl die gleichen Operatoren als auch die gleiche Diversifizierungsmethode (Shaking) verwendet. Auch die Initiallösung wird in Analogie zu $VNS \times SimA_{CPU}$ ermittelt. In Algorithmus 8 ist das Verfahren als Pseudocode dargestellt.

Der wesentliche Unterschied zur $VNS \times SimA_{CPU}$ ist das Annahmekriterium eines Moves. Wo zuvor ein Move, der die aktuelle Lösung verschlechtert hat, nur mit einer bestimmten Wahrscheinlichkeit angenommen wird, wird er in der TS_{CPU} sofort angenommen, sofern er nicht tabu ist. Im Rahmen der TS wird der beste Move auf jeden Fall angenommen - unabhängig von der Höhe der Verschlechterung. Das führt jedoch dazu, dass gerade bei Operatoren, die eine große Veränderung (oder Zerstörung) der aktuellen Lösung bewirken, ein Move trotzdem ausgeführt wird, der in der $VNS \times SimA_{GPU}$ im Rahmen der Akzeptanzwahrscheinlichkeit abgefangen werden kann. Aus dieser zerstörten Lösung schafft es die TS nicht, wieder in bessere Bereiche des Lösungsraumes zu gelangen, da gerade dann, wenn sie sich wieder etwas verbessert hat, dieser „zerstörerische“ Operator ausgeführt wird. Insbesondere hat sich das bei Cross-Exchange-Operatoren mit x-3,4,5 und 3,4,5-x bemerkbar gemacht, weshalb diese nicht mehr als Nachbarschaftsstruktur Anwendung finden. Alles andere findet in Analogie zur $VNS \times SimA_{CPU}$ statt.

Ein weiteres Differenzierungskriterium befindet sich in der Zurücksetzung der aktuellen Lösung auf die beste gefundene Lösung im Suchverlauf, wenn innerhalb einer bestimmten Anzahl von Iterationen keine neue beste Lösung gefunden wurde. Auch das Shaking kommt im Rahmen der TS nur zu bestimmten Zeiten zum Einsatz.

Die Verwaltung der Tabuliste findet auf CPU-Seite statt. Diese wird jedes Mal, wenn sie zur Bewertung der Moves benötigt wird, auf die Grafikkarte kopiert. Der Check bzw. der Aufbau der Tabuliste ist in

[84] Das begründet auch den Verzicht eines adaptiven Abkühlungsplans im Rahmen der $VNS \times SimA_{CPU}$, da in diesem Fall die Temperaturverwaltung auf der CPU stattgefunden hat und auf die GPU-Ausführung keine Auswirkung hat.

Algorithmus 8: Pseudocode der TS_{CPU}

```
1  Wähle Nachbarschaftsstruktur N_M, M = 1, ..., M_max
2  Setze Initiallösung s ∈ S
3  Wähle Abbruchbedingung
4  s' = s
5  while Abbruchbedingung nicht erfüllt do
6      M = 1
7      while M ≤ M_max do
8          bestimme besten Move aus Nachbarschaft N_M(s'), der nicht tabu ist
9          s' = wende besten Move an
10         if f(s') < f(s) then
11             s = s'
12         end
13         M = M + 1
14         Update Tabuliste (inkl. Schranken)
15         if Rücksetzen auf beste Lösung then
16             s' = s
17         end
18     end
19     if Shaking durchführen then
20         Shaking
21     end
22 end
```

Abschnitt 3.1.4 beschrieben. In Abbildung 3.9 sind die Schritte, die pro Thread zur Bewertung einer Erzeugerkante durchlaufen werden, aufgezeigt. Hier wird nun im Unterschied zur $VNS \times SimA_{CPU}$ die Prüfung der Tabuliste mit aufgenommen. Wichtig ist hierbei, dass vor der Prüfung der Tabuliste die Distanzveränderung bzw. die Distanz der resultierenden Lösung durch den Move ermittelt wird, da sie die Grundlage für die Prüfung des Tabustatus bildet. Es wird also die Distanzveränderung ermittelt, danach wird geprüft, ob der Move überhaupt noch in Frage kommt. Sollte der Move tabu sein, wird das mit einer hinreichend großen Zahl im Ergebnisarray - ebenso wie bei der Nichterfüllung bestimmter Kriterien - vermerkt. Sollte er nicht tabu sein, wird die neue Kapazität ermittelt und alles im Ergebnisarray gespeichert. Das wiederum bildet dann die Grundlage zur Bestimmung des besten Moves.

3.3.2 GPU-Variante: TS_{GPU}

Auch die GPU-Variante TS_{GPU} ist sehr ähnlich zur $VNS \times SimA_{GPU}$ aufgebaut. Auch hier besteht der wesentliche Unterschied in der Annahme eines Moves. Dabei gelten die gleichen Annahmen bzw. Aussagen, die in vorangegangenem Unterabschnitt getroffen wurden. So wird ebenfalls auf Teile des Cross-Exchange-Operators verzichtet. Der Ablauf entspricht weitestgehend dem aus Abbildung 3.7, wobei als Metaheuristik auf der GPU die TS zum Einsatz kommt.

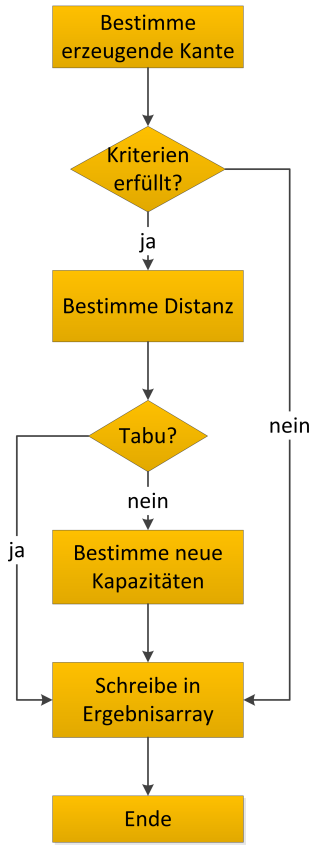


Abbildung 3.9: Aufgabe der Threads in der TS_{CPU}

Auch hier bearbeitet jeder gestartete Block separat eine Lösung. In Abbildung 3.10 ist das Vorgehen eines Blocks visualisiert. Die beiden wesentlichen Unterschiede finden sich in der Bewertung der erzeugenden Kanten, bei der nun die Tabuliste mit berücksichtigt wird und bei der Ausführung des besten Moves. Das bedeutet der beste Move, der unter Berücksichtigung des Tabustatus ermittelt wird, wird ohne weitere Abfrage auf die aktuelle Lösung angewendet. Nachdem der Move ausgeführt ist, muss die Tabuliste aktualisiert werden, d.h., es wird ein alter Wert entfernt und ein neuer hinzugefügt.

Das implementierte Programm ist so aufgebaut, dass wieder ein Großteil der Daten im SMem liegt. Diese Daten werden kurz vor Beendigung des Kernels in den globalen Speicher kopiert, damit sie bei einem erneuten Kernelaufruf wieder zur Verfügung stehen. Das gilt insbesondere für die Tabuliste, da jeder Block seine eigene verwaltet. Diese Verwaltung findet während des Kernelaufrufs im SMem statt. Durch Beenden des Kernels lässt sich in einem späteren erneuten Kernelaufruf nicht mehr auf diese Tabuliste zugreifen. Deshalb wird sie vor Abschließen des Kernels in den globalen Speicher und unmittelbar nach dem Start aus dem Globalspeicher in das SMem geschrieben, sodass die Tabuliste mehrere Kernelaufrufe überdauern kann. Die unterschiedlichen Tabulisten pro Block sorgen dafür, dass die Suchverläufe der Blöcke, wenn sie von einer gemeinsamen Lösung starten, nicht identisch sind bzw. sein müssen.[85] Des Weiteren sind die Längen der Tabulisten der Blöcke nicht zwangsläufig identisch, wodurch sich weitere Unterschiede in der parallelen Abarbeitung ergeben.

Auch wenn die beiden Verfahren $VNS \times SimA_{GPU}$ und TS_{GPU} große Gemeinsamkeiten aufweisen,

[85] Auch das Shaking-Verfahren sorgt durch seine stochastische Komponente für einen nicht identischen Suchverlauf. Dennoch wird dieser durch unterschiedliche Tabulisten weiter begünstigt.

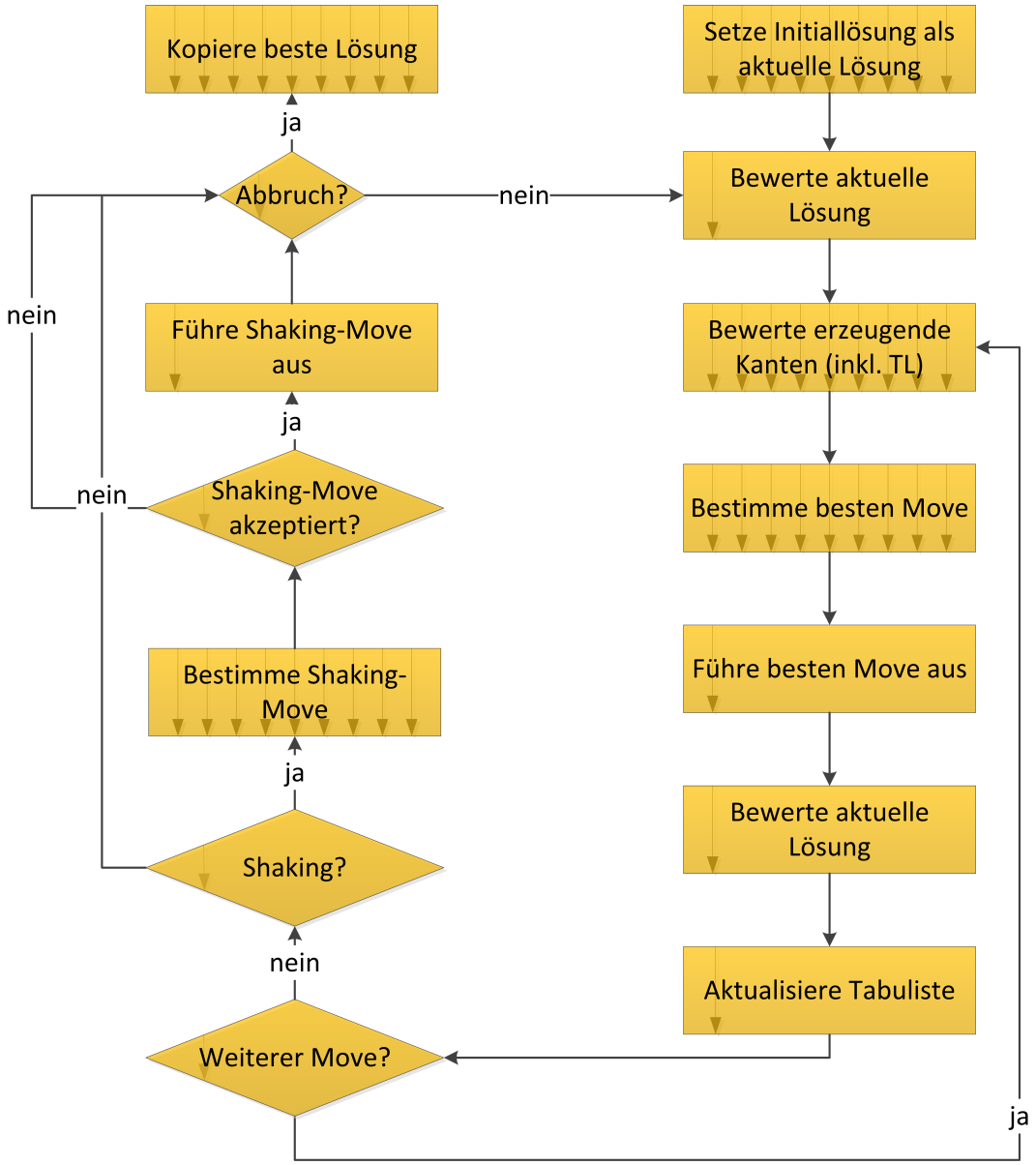


Abbildung 3.10: Ablauf eines Blocks in der TS_{GPU}

sind doch einige sehr einflussstarke strukturelle Unterschiede vorhanden, die mittels einer Performanceanalyse - sowohl hinsichtlich der Lösungsgüte als auch hinsichtlich der Laufzeit - aufgezeigt werden sollen.

3.4 Umsetzung eines GA

Die einzige Kommunikation, die in den Algorithmen $VNS \times SimA_{GPU}$ und TS_{GPU} zum Einsatz kommt, ist der Austausch der besten gefundenen Lösung. Das bedeutet, entweder alle Blöcke bauen auf der bisher besten gefundenen Lösung auf oder jeder Block arbeitet mit seiner aktuellen Lösung weiter. Hier stellt sich die Frage, ob nicht ein weniger starker Informationsaustausch, d.h. von Lösungsteilen zu weiteren Steigerungen in der Lösungsgüte herangezogen werden sollte. Dazu wird ein Cooperative-Simulated-Annealing-ähnliches (COSA) Shaking-Verfahren für die $VNS \times SimA_{GPU}$ implementiert,

das im nächsten Abschnitt erläutert werden soll, bevor dann ein Crossover, der sowohl im Kontext einer $VNS \times SimA_{GPU}$ als auch einer TS_{GPU} Anwendung findet, vorgestellt wird.

3.4.1 Shaking mit Cooperative-Simulated-Annealing (COSA)

Für COSA wird auf die grundlegende Idee von Wendt (1995) zurückgegriffen, die darin besteht, dass zur Anwendung von Operatoren im SimA andere Individuen als Informationsspender dienen, um daraus den Move, der ausgeführt wird, zu bestimmen. Um das Vorgehen zu verdeutlichen, soll in Anlehnung auf das Beispiel von Wendt (1995, S. 153) Bezug genommen werden. Dabei ist eine TSP-Lösung (1 2 6 3 5 4) gegeben und die Kante (2 6) soll entfernt werden. Um zu bestimmen, welche Kante nun in die Lösung eingefügt wird, wird ein anderes Individuum (4 1 3 2 5 6) befragt. Hier zeigt sich, dass 2 mit 5 verbunden ist, wodurch nun verlangt wird, dass in der neuen Lösung ebenfalls Stadt 2 mit 5 verbunden ist. Dadurch ergäbe sich im Falle des Relocate-Operators, dass Stadt 5 zwischen Stadt 2 und 6 eingefügt werden muss. Daraus resultiert folgende neue Lösung (1 2 5 6 3 4). Dieses Vorgehen lässt sich mit allen in dieser Arbeit verwendeten Operatoren durchführen. Es wird immer geprüft, welchen Nachbar ein bestimmter Knoten hat und somit eine erzeugende Kante ermittelt.

COSA soll im Rahmen der Shaking-Prozedur Anwendung finden. Das Shaking-Verfahren in der $VNS \times SimA_{GPU}$ bestimmt sozusagen eine zufällige Kante und bewegt dann in Abhängigkeit der Akzeptanzwahrscheinlichkeit die aktuelle Lösung in diese Richtung. Nun wird im Shaking-Verfahren in Analogie zu oben gezeigtem Beispiel eine erzeugende Kante selektiert. Dazu ist es notwendig, dass alle Blöcke auf die Lösungen der anderen Blöcke zugreifen können. Das ist lediglich über den Weg des globalen Speichers zu erreichen. Dazu schreiben alle Blöcke zu bestimmten, aber nicht unbedingt gleichen Zeitpunkten ihre aktuelle Lösung in den globalen Speicher. Ist nun ein Block in seiner Shaking-Methode angelangt, holt er sich zufällig die Tour eines anderen Blocks und ermittelt auf Basis dieser die erzeugende Kante. Basis bildet dabei ein zufällig gewählter Knoten auf seiner eigenen Route. Mit diesem wird der nächste Nachbar des Knoten im Informationsspender ermittelt und somit die erzeugende Kante festgelegt. Hierbei ist es nicht notwendig, dass eine Synchronisation zwischen den Blöcken stattfindet, da es für das Shaking-Verfahren weniger relevant ist, ob es mit der Lösung, die vor einer Iteration oder der Lösung, die vor 20 Iterationen in dem Nachbarblock erzeugt wurde, arbeitet. Es soll schließlich nur eine grobe Richtung vorgegeben werden, in die die aktuelle Lösung bewegt wird.

Nachdem die Erzeugerkante bestimmt ist, wird geprüft, ob der Operator, der die Basis für die neue Lösung stellt, zu einer gültigen neuen Lösung führt. Ist dies der Fall, wird die aktuelle Lösung dem Operator unterzogen, andernfalls wird in dieser Shaking-Iteration kein Move durchgeführt. Hier findet keine weitere Prüfung des Moves mittels einer Akzeptanzwahrscheinlichkeit statt. Der Verzicht lässt sich dadurch begründen, dass durch die Bestimmung der Erzeugerkante bereits eine starke Einschränkung für potentielle Diversifizierungsmoves stattfindet. In Tests hat eine weitere Auswahl basierend auf einer Akzeptanzwahrscheinlichkeit gezeigt, dass keine Moves im Shaking-Verfahren ausgeführt bzw. die Temperatur so hoch gesetzt werden muss, dass sich ein Verzicht auf die Wahrscheinlichkeit förmlich aufdrängt. Als Shaking-Operatoren kommen wie zuvor die unterschiedlichen Cross-Exchange-Konfigurationen zum Einsatz.

Das grundsätzliche Ziel des COSA ist es, Sprünge im Lösungsraum zu vermeiden und stattdessen die Individuen aufeinanderzubewegen (Wendt, 1995, S. 153). In der hier vorgenommenen Implementierung wird sich lediglich die Idee des Informationsspenders zu Nutze gemacht. Durch die damit festgelegte

Erzeugerkante kann es zu großen Sprüngen im Shaking-Verfahren kommen. Im ursprünglichen COSA ist es möglich, diese durch die Akzeptanzwahrscheinlichkeit zu vermeiden. Auch wenn es nur teilweise Ähnlichkeiten zu dem Original-COSA gibt, soll auch weiterhin im Rahmen dieses Beitrags von COSA gesprochen werden. Gemeint ist dann jedoch die hier verwendete Version. Weiterhin muss berücksichtigt werden, dass COSA im Shaking nur einmal versucht eine gültige Erzeugerkante zu ermitteln. Dadurch sinkt die Wahrscheinlichkeit, dass ein Shaking-Move gefunden und ausgeführt wird. Um dem zu begegnen, wird, wie oben beschrieben, auf die Annahmewahrscheinlichkeit verzichtet. Allerdings wäre es hier in Analogie zu dem Basis-Shaking-Verfahren auch denkbar, COSA mehrere Möglichkeiten zu geben, um eine erzeugende Kante zu ermitteln.

3.4.2 Crossover

Wie schon beschrieben sollen durch die Anwendung eines Crossover mehr Informationen zwischen den verschiedenen Blöcken ausgetauscht werden. Um das zu erreichen, wird wie im Multistart-Ansatz ein Lösungspool auf CPU-Seite verwaltet, den man ebenso als Population ansehen kann, weshalb diese beiden Begriffe im Folgenden synonym Verwendung finden.

Die Anzahl der Lösungen im Lösungspool entspricht der Anzahl der Blöcke, die gestartet werden. Die CPU entscheidet dann, welche Lösung von welchem Block bearbeitet werden soll. Sie kann somit großen Einfluss auf die Lösung, die zur GPU gesendet wird, nehmen. Daher ist es naheliegend, den Crossover CPU-seitig zu implementieren.

In Abbildung 3.11 ist das grobe Vorgehen skizziert. Die Generierung der Ausgangsbasis findet wie bisher mittels des Frameworks von Groër et al. (2010) statt, wodurch unterschiedliche Lösungen garantiert werden können. Die Methode, die zur Verteilung der Lösungen zum Einsatz kommt, entscheidet, ob entweder alle Blöcke das gleiche und damit das bisher beste gefundene Resultat bearbeiten oder, ob jeder Block genau eine (unterschiedliche) Lösung als Ausgangsbasis erhält. Schließlich führt die GPU ihre Berechnungen durch, wobei es hierbei unerheblich ist, ob die $VNS \times SimA_{GPU}$ oder TS_{GPU} zum Einsatz kommt, d.h., an dieser Stelle können die Verfahren beliebig ausgetauscht werden. Nachdem die GPU ihre Arbeiten abgeschlossen hat, werden, wie bereits erläutert, die Lösungen von der GPU eingesammelt und entschieden, welche in die Population einfließen. Es ist also zu bestimmen, welche neuen Lösungen die alten ersetzen. Hierbei muss darauf geachtet werden, dass sich die Größe einer Population nicht ändert, da von einer konstant großen Population ausgegangen wird und jedem Block genau eine Lösung zur weiteren Bearbeitung zugewiesen werden kann.[86] Wenn also eine neue Lösung hinzugefügt werden soll, muss eine alte Lösung durch sie ersetzt werden. Nachdem alle optimierten Lösungen eingesammelt sind, ist zu entscheiden, ob sie einem Crossover unterzogen werden. Wenn ein Crossover ausgeführt wird muss der Lösungspool wieder entsprechend aktualisiert und darauf geachtet werden, dass seine Größe nicht ansteigt. Schließlich werden die Lösungen wieder auf die GPU verteilt und die nächste Iteration startet.

Bei dem in der Arbeit implementierten Crossover handelt es sich um eine recht einfache Variante des Kundenaustauschs. Es kommen zwei Eltern zum Einsatz. Dabei spielt jedes Individuum in der Population mindestens einmal eine Elternrolle. Das heißt, während des Crossovers ist jedes Individuum

[86] Es ist ebenso denkbar, dass eine größere Population vorgehalten wird. Allerdings wird dieses Vorgehen in der vorliegenden Arbeit nicht weiter verfolgt, um ein 1:1-Mapping der Anzahl der Lösungen, die vorgehalten werden, auf die Anzahl der SMs der GPU zu erreichen.

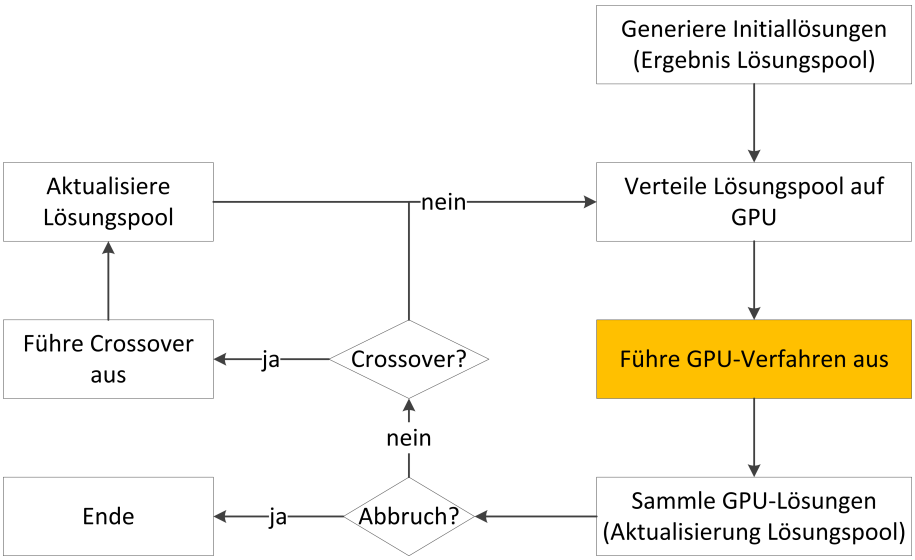


Abbildung 3.11: Flussdiagramm des implementierten GAs

als Elternteil einmal gesetzt. Der andere Elternteil ergibt sich nach der Roulette-Wheel-Selektion, die dafür sorgt, dass Individuen mit einer höheren Qualität - sprich Lösungsgüte - bevorzugt werden. Der Crossover wählt nun zufällig eine Kundensequenz[87] aus einem Elternteil aus und fügt ihn in dem anderen Elternteil ein. Die Einfügestelle ergibt sich aus dem nächsten Nachbarn des ersten Kunden in der einzufügenden Kundensequenz. Kunden, die bereits an anderer Stelle bedient werden, werden von dieser Stelle entfernt, sodass weiterhin garantiert ist, dass jeder Kunde nur einmal angefahren wird. Es handelt sich also um einen OX (vgl. Wendt (1995, S. 81f.)). Die so erzeugte neue Lösung ersetzt die alte Lösung, die als Ausgangsbasis diente, und wird nun von dem GPU-Verfahren optimiert. Allerdings wird dafür gesorgt, dass die beste Lösung, die in der Population enthalten ist, erhalten bleibt.

Beim Einsatz des Crossovers ist die Wahrscheinlichkeit, dass eine ungültige Lösung generiert wird, sehr hoch. Deshalb enthält das Verfahren noch einen Reparaturmechanismus, der dafür sorgt, dass gültige Lösungen geliefert werden. Der Mechanismus durchläuft die erzeugte Lösung und prüft, ob eine Kapazitätsverletzung vorliegt. Wenn dies der Fall ist, wird an der Stelle, an der erstmals eine Verletzung auftritt, ein weiterer Depotknoten eingefügt und damit die Restriktionsverletzung geheilt. Dieses Vorgehen sorgt dafür, dass die resultierenden Lösungen i.d.R. eine höhere Anzahl von Routen aufweisen als die Elternteile. Das spielt jedoch nur eine untergeordnete Rolle, da diese Routen zumeist im Verlauf des GPU-Verfahrens wieder wegoptimiert werden - sofern damit eine geringere Distanz erzielt wird.

Nun stellt sich noch die Frage, wie das Verfahren eingeordnet werden kann. Die Beschreibung verdeutlicht, dass fundamentale Bestandteile, die grundsätzlich in einem GA enthalten sind, hier fehlen. Zu nennen ist bspw. der Mutationsoperator, der für eine Diversifizierung sorgen soll. Allerdings kann man in Analogie zu Prins (2004, S. 1991) den Mutationsoperator ebenso als Lokalsuche, wie sie von der GPU realisiert wird, ansehen. Das widerspricht zwar einerseits dem Gedanken, dass zusätzliche Diversifizierung durch den Mutationsoperator erreicht werden soll. Andererseits wird gerade durch den hier implementierten Crossover die Lösung so stark verändert, dass damit nicht zwangsläufig eine weitere Diversifizierung notwendig ist. Die Kombination von GAs und Lösungsverfahren, die auf

[87] Hier ist es wichtig zu beachten, dass die Sequenz keinen Depotknoten enthalten darf.

Lokalsuchen basieren, werden oftmals als hybride GAs oder memetische Algorithmen (MA) bezeichnet (Moscato und Cotta, 2010, S. 162). MAs versuchen dabei sämtliches Wissen über das zu lösende Problem auszuschöpfen (Moscato und Cotta, 2010, S. 141). Hierbei verwenden sie i.d.R. populationsbasierte Verfahren wie GAs und verknüpfen diese mit Lösungsverfahren, die auf Lokalsuchen basieren.

Man könnte auch argumentieren, dass der Crossover ein Bestandteil der $VNS \times SimA_{GPU}$ bzw. TS_{GPU} ist und in den Verfahren die Aufgabe hat, für eine größere Exploration des Suchraums zu sorgen. Das bedeutet, in den Verfahren ist bereits eine Diversifizierungsmethode, nämlich das Shaking, enthalten. Nun wird diese nach einer bestimmten Anzahl Iterationen erweitert, sodass eine weitergehende Diversifizierung stattfindet. Trotzdem soll sich für die Einordnung in die ursprüngliche GA-Notation entschieden werden und dem Vorgehen Prins (2004), der die Mutation als Lokalsuche definiert, gefolgt werden.

3.5 Einordnung und Vergleich der Implementierungen

Nachdem alle implementierten Heuristiken bzw. Metaheuristiken vorgestellt sind, sollen sie zunächst anhand des Schemas von Crainic und Toulouse (2010), das in Kapitel 2.4.1.2 vorgestellt wird, eingeordnet werden.

Die beiden CPU-Varianten, d.h., die Verfahren, die viele Berechnungen auf der CPU durchführen, sind im Rahmen der SCC als 1C anzusehen. Es existiert dabei ein Prozess, der die einzelnen Threads, die die Moves bewerten bzw. die den besten ermitteln, steuert und die Ergebnisse von ihnen an einer Stelle einsammelt. Bei Betrachtung der SCaC-Dimension muss man das parallele Verfahren unter RS einordnen, da die einzelnen Threads jeweils zu bestimmten Zeiten ihre Ergebnisse fertigstellen und die Ermittlung des besten Moves erst dann startet, wenn alle Threads beendet sind. Ein Informationsaustausch zwischen den Threads findet nicht statt. Auf der Ebene der SD sind die Verfahren unter SPSS einzuordnen, da nur mit einer Lösung gearbeitet und im Suchverlauf nur je ein Verfahren verwendet wird.

Im Rahmen der $VNS \times SimA_{GPU}$ ändert sich das Bild ein wenig. Auch hier ist nur ein Prozess an der Steuerung der Kommunikation beteiligt, weshalb man das Verfahren als 1C einordnen sollte. Bei Betrachtung der SCaC-Dimension ist das Verfahren jedoch in die CO-Klasse einzuteilen. Die einzelnen Threads bzw. Blöcke können grundsätzlich, ohne auf die anderen Blöcke warten zu müssen, arbeiten. Es findet jedoch eine Synchronisation statt, wenn alle Lösungen der einzelnen Blöcke eingesammelt werden, weshalb es hier durchaus denkbar wäre, das Verfahren ebenfalls in RS einzuordnen. Allerdings überwiegt die Unabhängigkeit der Threads und Blöcke, weshalb sich für eine Einordnung in CO entschieden wird. Da nur Lösungen ausgetauscht werden, findet auch hier kein Wissenstransfer statt. Im Rahmen der SD handelt es sich bei dem Algorithmus um ein Multistart-Verfahren mit jeweils unterschiedlichen Startlösungen aber grundsätzlich gleichen Suchstrategien, weshalb es in die Klasse der MPSS einzuordnen ist. Die gleiche Kategorisierung kann im Rahmen der TS_{GPU} gemacht werden. Auch bei Betrachtung der $VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan kommt man zu einer ähnlichen Einordnung. Allerdings muss man hier die Frage stellen, ob es sich nicht auch um unterschiedliche Suchstrategien handelt. Hintergrund dessen ist die Tatsache, dass hier jeder Suchblock seine eigene Temperatur je nach Niveau einstellt und es somit zu sich stark unterscheidenden Suchverläufen kommen kann. Deshalb soll die $VNS \times SimA_{GPU}$ mit adaptiver Abkühlung im Rahmen der SD der Klasse MPDS zugeordnet werden. An dieser Stelle kann man das Argument aufführen, dass gleiches im Rahmen

der TS gilt, wenn unterschiedliche Längen der Tabuliste verwendet werden. Allerdings wird im Rahmen der TS die Länge der Tabuliste vergleichsweise gering - insbesondere bei großen Instanzen - variiert, sodass von einer Einordnung in die MPDS-Klasse abgesehen wird.

Bei Betrachtung der $VNS \times SimA_{GPU}$ mit COSA ändert sich im Vergleich zur Version ohne COSA lediglich die Ebene der SCaC. Neben dem Austausch der besten Lösung kommen nun auch Informationsspender zum Tragen. Das heißt, hier werden mehr Informationen geteilt als nur Lösungen.[88] Weiterhin findet der Informationsaustausch zwischen den Threads zu unterschiedlichen Zeitpunkten und nicht synchronisiert statt, weshalb die $VNS \times SimA_{GPU}$ mit COSA der Klasse KC zuzuordnen ist.

Auch im Kontext der GAs, die in der Arbeit implementiert werden, findet die Suchkontrolle bzw. Steuerung von einem Prozess aus statt, weshalb auch hier die Klasse 1C anzusetzen ist. Beim Informationsaustausch muss man differenzieren. Es werden durch den Crossover Lösungskomponenten ausgetauscht, die stets zu festen Zeitpunkten stattfinden. Deshalb muss, obwohl die Blöcke unabhängig voneinander arbeiten, die Klasse KS verwendet werden. Hier zeigt sich eine Schwäche bei Nutzung dieses Schemas. Im Hinblick auf die Version, die zusätzlich COSA verwendet, ist nun nicht mehr klar, ob das Verfahren in KS wegen dem Crossover oder in KC wegen COSA eingeordnet werden sollte. Da grundsätzlich der Crossover bzw. der Zeitpunkt, wann der Crossover stattfinden soll, den Takt vorgibt, soll sich für diese Einordnung entschieden werden. Im Rahmen der SD muss unterschieden werden. Dabei benutzen die $VNS \times SimA_{GPU}$, $VNS \times SimA_{GPU}$ mit COSA und TS_{GPU} unterschiedliche Startlösungen, verwenden aber grundsätzlich die gleiche Suchstrategie, weshalb sie in MPSS eingeordnet werden. Demgegenüber steht wieder das Verfahren mit der adaptiven Abkühlung, das in MPDS eingeordnet wird. Auch hier wird die Problematik des Schemas ersichtlich. Dennoch sollte es zum Vergleich bzw. zur groben Einordnung der Verfahren gute Einblicke ermöglichen. Tabelle 3.4 enthält die Daten nochmals aggregiert.

Verfahren	SCC	SCaC	SD
$VNS \times SimA_{CPU}$	1C	RS	SPSS
TS_{CPU}	1C	RS	SPSS
$VNS \times SimA_{GPU}$	1C	CO	MPSS
$VNS \times SimA_{GPU}$ mit adaptiver Abkühlung	1C	CO	MPDS
$VNS \times SimA_{GPU}$ mit COSA	1C	KC	MPSS
TS_{GPU}	1C	CO	MPSS
GA mit $VNS \times SimA_{GPU}$	1C	KS	MPSS
GA mit $VNS \times SimA_{GPU}$ mit adaptiver Abkühlung	1C	KS	MPDS
GA mit $VNS \times SimA_{GPU}$ mit COSA	1C	KS	MPSS
GA mit TS_{GPU} mit COSA	1C	KS	MPSS

Tabelle 3.4: Einordnung der Metaheuristiken nach dem Schema von Crainic und Toulouse (2010)

Versucht man die Algorithmen nach Art der Zerlegung einzuordnen, fällt auf, dass dies nur weniger trennscharf möglich ist. Im Falle der $VNS \times SimA_{CPU}$ und TS_{CPU} bzw. der dort verwendeten Operatoren liegt eindeutig eine Datenzerlegung vor, da die gleiche Funktion auf viele unterschiedliche Daten angewendet wird. Auch bei $VNS \times SimA_{GPU}$ (mit COSA), TS_{GPU} und GA mit den beiden

[88] Auch wenn die Informationen auf Lösungen basieren.

erstgenannten liegt eine Datenzerlegung vor, die auf der Ebene der Lösungen vonstatten geht, aber grundsätzlich gleich abläuft. Allerdings könnte man im Falle der TS_{GPU} auch argumentieren, dass durch die unterschiedliche Länge der Tabulisten entsprechend andere Funktionen angewendet werden, weshalb es sich hierbei um eine Kombination von funktionaler Zerlegung und Datenzerlegung handelt. Gleiches kann man im Rahmen des adaptiven Abkühlungsplans argumentieren, da hier unterschiedliche Temperaturniveaus verwaltet werden.

Bei Betrachtung der Granularität lassen sich die beiden CPU-Verfahren als feingranular einordnen und alle anderen Ansätze als grobgranular. Gleichzeitig ist der Grad der Parallelität bei Betrachtung entsprechend großer Instanzen bei $VNS \times SimA_{CPU}$ und TS_{CPU} höher als bei den GPU-Varianten. Bei letzteren werden konstant - unabhängig von der Instanzgröße - $16 \cdot 512 = 8192$ (vgl. Abschnitt 4.2.2.1) Threads bzw. Teilaufgaben gestartet. Bei der CPU-Variante wird hingegen in Abhängigkeit der Anzahl der Generatorkanten die Anzahl der gestarteten Teilaufgaben determiniert. Das heißt, selbst bei einer kleinen Instanz mit 100 Knoten liegen über 10000 Erzeugerkanten vor, weshalb im Rahmen der CPU-Variante ein höherer Grad der Parallelität im Vergleich zur GPU-Variante vorliegt.

Parallelen zur Arbeit von Schulz (2013) lassen sich nur schwer ziehen, da die Implementierung des Autors auf einer grundsätzlich anderen Idee basiert. Hierbei wird immer die vollständige Nachbarschaft abgesucht, wohingegen die Umsetzung dieser Arbeit so ausgelegt ist, dass durch wenig Aufwand die Nachbarschaft erheblich verkleinert werden kann, indem auf das Konzept von Toth und Vigo (2003) zurückgegriffen wird und lange Kanten, die wahrscheinlich nicht einer guten Lösung angehören, aus der Menge der erzeugenden Kanten entfernt werden können. Demgegenüber können die Ergebnisse von Luong et al. (2011, 2013) bestätigt werden, indem auch in dieser Arbeit die schnellste Implementierung jeweils auf den Texturspeicher zum Ablegen der Distanzmatrix zurückgreift (vgl. Abschnitt 4.1).

Ähnlichkeiten zu der Implementierung von Coelho et al. (2012) sind ebenfalls ersichtlich. So werden in der Umsetzung der Arbeiten ebenfalls zuerst alle möglichen Moves der Nachbarschaften bewertet und in eine Ergebnismatrix geschrieben. Diese wird im Anschluss nach der besten Lösung durchsucht, wobei hier die CPU zum Einsatz kommt. Das bedeutet, bei Betrachtung der Operatoren liegt grundsätzlich der gleiche Aufbau wie in dieser Arbeit zugrunde, mit dem Unterschied, dass in der vorliegenden Arbeit der beste Move ebenfalls mit der GPU bestimmt wird und die Menge an Daten, die transferiert werden muss, reduziert werden kann. Ähnlich ist die Implementierung von Weyland et al. (2013), bei der ebenfalls die Nachbarschaft bzw. die Zielfunktion von der GPU evaluiert wird.

In der Arbeit von Jin et al. (2012) wird eine TS eingesetzt, bei der jeweils mehrere TS parallel auf einer CPU gestartet werden. Analog dazu werden in diesem Beitrag mehrere parallele Blöcke mit jeweils einer TS auf der GPU gestartet (vgl. TS_{GPU}). Gleiches gilt für die Publikationen von Jin et al. (2011) sowie Cordeau und Maischberger (2012), die ebenfalls mehrere Threads mit eigenen TS starten. In Analogie zu Cordeau und Maischberger (2012) wird in der hier vorliegenden Arbeit ebenfalls mit mehreren Initiallösungen gearbeitet.

Ein ähnlicher schematischer Aufbau wie in dieser Arbeit stammt aus dem Beitrag von Barbucha (2011). Der Autor verwendet Agenten, die unabhängig voneinander Lösungen des CVRPs optimieren. Auch in dem hier vorliegenden Beitrag könnte man die Blöcke, die Lösungen parallel verarbeiten, als mehr oder weniger unabhängige Agenten ansehen, die von Zeit zu Zeit wie in der Publikation von Barbucha (2011) ihre gefundenen Lösungen austauschen.

In der Arbeit von Garcia et al. (1994) findet die Parallelisierung durch die Verteilung der Knoten

auf die beteiligten Prozessoren statt. Mittels dieser werden die Moves determiniert, die ein Prozessor bewerten muss. Hierbei handelt es sich demnach um ein ähnliches Vorgehen wie in der vorliegenden Arbeit. Der Algorithmus von Garcia et al. (1994) ist jedoch grob-granularer, da ein Prozessor mehrere Knoten betrachtet.

3.6 Zusammenfassung

In der Arbeit kommen insgesamt sechs verschiedene lokale Suchoperatoren zum Einsatz. Es handelt sich um den Relocate-, Swap-, Or-Opt-, 2-Opt-, 2-Opt*- sowie Cross-Exchange-Operator. Letzterer deckt Teile der anderen Operatoren ab, wobei die anderen Operatoren trotzdem separat implementiert sind, um spezifische Eigenschaften - insbesondere bei Bestimmung der Kapazitäten - besser ausnutzen zu können und Performancevorteile zu erzielen. Alle Operatoren werden sowohl in einer CPU- als auch einer GPU-Variante umgesetzt und ermöglichen eine direkte Aussage über die Skalierungsfähigkeiten der unterschiedlichen Hardwarearchitekturen (siehe hierzu auch Abschnitt 4.1).

Die Operatoren finden in einer VNS, die Bestandteile eines SimA enthält, Anwendung ($VNS \times SimA_{CPU}$). Dabei werden große Teile wie z.B. die Bewertung der Gesamttour von der CPU übernommen, wohingegen die GPU lediglich die Bewertung sowie die Bestimmung des besten Moves vornimmt. Um die Grafikkarte besser auszulasten, werden weitere Teile des Algorithmus in der $VNS \times SimA_{GPU}$ auf die Grafikkarte ausgelagert und es somit ermöglicht, dass sie theoretisch vollkommen autark Probleme lösen kann. Um dabei die Nutzung der Hardwareressourcen weiter auszubauen, werden mehrere Lösungen parallel bearbeitet, sodass es sich schließlich um ein Multistart-Verfahren handelt.

Ähnlich dazu wird bei der Umsetzung einer TS vorgegangen, d.h., auch hier berechnet die Basisversion TS_{CPU} vergleichsweise „wenig“ auf der GPU, wohingegen die Grafikkarte bei der erweiterten Version TS_{GPU} bedeutend mehr Aufgaben übernimmt und ebenso vollkommen autark als Multistart-Verfahren arbeiten könnte.

Im Rahmen der VNS und SimA werden weiterhin Komponenten eingebaut, die oftmals erwünscht sind und bei denen im Rahmen der numerischen Studien vor allem die Performance-Auswirkungen im Vordergrund stehen. Dabei wird einerseits ein adaptiver Abkühlungsplan in die $VNS \times SimA_{GPU}$ implementiert, der ebenfalls vollständig von der GPU verwaltet wird. Andererseits wird die Diversifizierungsmethode bzw. das Shaking-Verfahren durch die Grundidee des COSA ersetzt.

In einem letzten Schritt werden beide Multistart-Verfahren $VNS \times SimA_{GPU}$ und TS_{GPU} im Kontext eines simplen GA eingesetzt. Hierbei übernehmen die $VNS \times SimA_{GPU}$
bzw. TS_{GPU} jeweils die Aufgabe bzw. treten jeweils an die Stelle des Mutationsoperators im Rahmen des GA.

4 Numerische Analysen der Implementierungen

Die folgenden Analysen werden - sofern nicht anders genannt - auf einem Rechner mit dem Betriebssystem Windows 7 64 Bit durchgeführt. Die Software wird mit C++ und dem CUDA-Toolkit 4.2 mit Visual Studio 2010 von Microsoft entwickelt. Weiterhin kommen in dem PC 4 GB Arbeitsspeicher und ein Intel Core i7 860 mit 2,8 GHz zum Einsatz. Der Treiber für die Grafikkarte Nvidia Geforce 580 GTX trägt die Revisionsnummer 301.42.

Zunächst sollen einzelne Operatoren näher analysiert werden, bevor auf Metaheuristiken, die diese Operatoren einsetzen, eingegangen wird. Die Reihenfolge resultiert aus der Notwendigkeit, zunächst Parameter für die Operatoren zu bestimmen, damit sie zufriedenstellende bzw. bestmögliche Ergebnisse im Hinblick auf die Rechengeschwindigkeit ermöglichen.

4.1 Performanceanalyse der lokalen Suchoperatoren

Dieser Abschnitt untersucht die Auswirkungen, die bei Nutzung einer GPU auftreten, bezogen auf lokale Suchoperatoren, die in den Abschnitten 2.3.2.2 bzw. 3.1 vorgestellt wurden.

4.1.1 Relocate-Operator

4.1.1.1 Auswirkungen des Speichertyps, der Thread- und der Knotenanzahl auf die Kernelzeit

Zunächst soll untersucht werden, in welchem Speicherbereich auf der GPU welches Datum abgelegt werden soll. Konkret dreht sich die Frage vornehmlich um den Speicher, in dem die Distanzmatrix abgelegt werden soll.[89]

Abbildung 4.1 zeigt in Abhängigkeit von der Thread- und Kundenanzahl[90] die Dauer eines Kernels zur Bewertung aller erzeugenden Kanten bei Anwendung des Relocate-Operators. Die Distanzen werden in diesem Fall jedes Mal, wenn sie benötigt werden, auf Grundlage der x- und y-Koordinaten, die im Globalspeicher abgelegt sind, neu berechnet. Die roten Punkte stellen die tatsächlichen Messpunkte dar und die graue Fläche ist die Anpassungsfläche, die auf Basis der Messpunkte generiert wird. Wie aus der Grafik ersichtlich, werden die Thread- und die Knotenanzahl in diskreten Schritten variiert. Die Knotenanzahl resultiert implizit durch die verwendeten Benchmarkinstanzen. In Tabelle 4.1 finden sich

[89] Alle anderen Daten, die vorgehalten werden müssen, kommen theoretisch nur zur Speicherung ins SMem in Frage. Durch das Zugriffsmuster, das auf diesen Speicher jedoch entstehen würde, käme es zu vielen sog. Bank-Konflikten, sodass sich die Speichervorteile wieder aufheben und keine bzw. nur sehr geringe Geschwindigkeitsvorteile gegenüber dem Globalspeicher erzielen lassen. Deshalb werden diese Möglichkeiten nicht weiter analysiert.

[90] Die Kunden- bzw. Knotenanzahl impliziert auch die Anzahl der Kanten, die betrachtet werden, da sich die Erzeugerkanten direkt daraus ableiten. Deshalb gelten die Verläufe, die im Folgenden skizziert werden auch für eine entsprechende Variation der Kantenzahl.

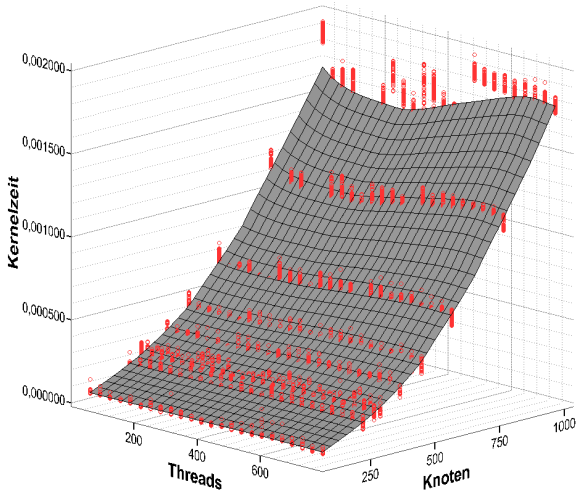


Abbildung 4.1: Kernelzeit des Relocate-Operators bei direkt berechneten Distanzen und Koordinaten im Globalspeicher

die verwendeten Instanzen für die folgenden Tests.[91] Die Threads werden so variiert, dass die Anzahl der Threads pro Block ein Vielfaches von 32 ist (32 - 768).[92] Pro Thread-Knoten-Kombination werden jeweils 100 Durchläufe durchgeführt.

Instanz	Knotenanzahl
Christofides et al. (1979) c3	101
Golden et al. (1998) g9	256
Golden et al. (1998) g10	324
Golden et al. (1998) g11	400
Golden et al. (1998) g12	484
Golden et al. (1998) g18	301
Gehring und Homberger (1999) R1_600	601
Gehring und Homberger (1999) R1_800	801
Gehring und Homberger (1999) R1_1000	1001

Tabelle 4.1: Benchmarkinstanzen zum Testen der lokalen Suchoperatoren

Aus Abbildung 4.1 kann man insbesondere bei einer großen Knotenanzahl erkennen, dass die Kernelzeit, die in dieser und den folgenden Abbildungen in Sekunden angegeben ist, bei einer geringen Threadanzahl höher ist als bei einer höheren Threadzahl. Dieser Umstand kehrt sich ins Gegenteil, wenn die Anzahl

[91] Die diskreten Schritte bei der Knotenanzahl ergeben sich unmittelbar aus den verwendeten Probleminstanzen. Es scheint in diesem Zusammenhang wenig sinnvoll, die Instanzen selbst zu generieren, um damit kontinuierlich die Knotenzahl variieren zu können, sondern es ist sinnvoller, bereits existierende Probleminstanzen zu verwenden. Die Benchmarkinstanzen werden detailliert in Abschnitt 2.5.2 erläutert.

[92] Wie in Kapitel 2.7.4 beschrieben, ist die Warp-Größe 32. Aufgrund dessen sollte aus Performancegründen darauf geachtet werden, dass die Threadzahl pro Block ein Vielfaches von 32 ist. Weiterhin wäre es theoretisch möglich, bis zu 1024 Threads pro Block zu starten. Durch die verwendeten Ressourcen (z.B. Register) in der vorliegenden Implementierung ist es jedoch nicht möglich, mehr als 768 Threads (bei einigen Verfahren sogar weniger) zu starten.

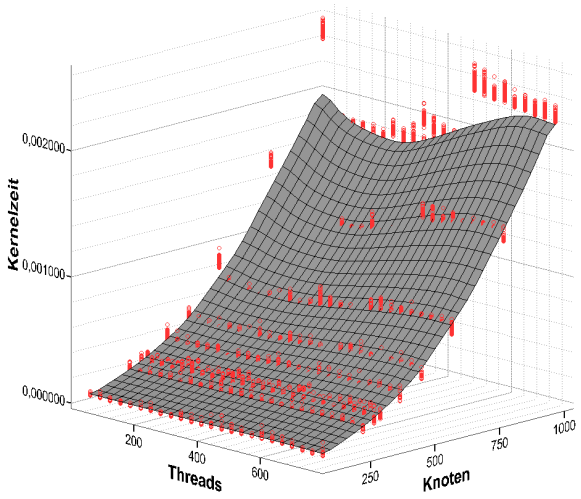


Abbildung 4.2: Kernelzeit des Relocate-Operators bei direkt berechneten Distanzen und Koordinaten in Constant Memory

der Threads weiter ansteigt. Das lässt sich damit begründen, dass bei einer Threadzahl von bspw. 192 bereits zwei weitere Blöcke zum Starten vorbereitet werden können, da die Ressourcen auf der Grafikkarte dafür ausreichen. Mit steigender Threadzahl wie bspw. 512 kann kein weiterer Block zur Ausführung vorbereitet werden, da somit 1024 aktive Threads auf dem Multiprozessor wären und das die verfügbaren Ressourcen bei der vorliegenden Implementierung übersteigt (vgl. oben). Bei einer sehr kleinen Threadanzahl ist es zwar auch möglich, dass weitere Blöcke für den Start vorbereitet werden, allerdings werden durch die geringe Anzahl die Vorteile, die sich durch die Cache-Strukturen der GPU ergeben, nicht vollständig genutzt, sodass damit der Vorteil der Vorbereitung mehrerer Blöcke wieder zunichte gemacht wird.

Bei der soeben betrachteten Implementierung werden die Koordinaten zur Berechnung der Distanzen im globalen Speicher vorgehalten und die Distanzen jedes Mal, wenn sie benötigt werden, neu berechnet. Nun stellt sich die Frage, inwiefern man dieses Vorgehen besser bzw. schneller gestalten kann, um die Kernelzeit zu verringern. In Abbildung 4.2 findet sich der Verlauf der Kernelzeit, wenn die Koordinaten im Constant Memory abgelegt sind und die Distanzen nach wie vor direkt berechnet werden. Man kann einen ähnlichen Verlauf wie zuvor erkennen, d.h., mit steigender Threadzahl sinkt die Ausführungsdauer bis zu einem gewissen Punkt, um dann wieder anzusteigen. Allerdings führt die Nutzung des Constant Memory zur Speicherung von Problemdaten zu einem Geschwindigkeitsverlust, was damit zu erklären ist, dass das Constant Memory sehr schnell ist, wenn alle Threads eines Blocks auf dasselbe Datum zugreifen. Sobald jedoch auf viele unterschiedliche Daten zugegriffen wird, müssen diese serialisiert werden, wodurch sich die Geschwindigkeitsvorteile aufheben (vgl. Nvidia (2012, S. 73), Nvidia (2011a, S. 46) und Luong et al. (2011, S. 329)). Genau diese unterschiedlichen Zugriffe auf Daten sind zur Bestimmung der Distanzen notwendig, weshalb es hier zu einer Verschlechterung der Laufzeit kommt.

Eine weitere Möglichkeit der Performanceoptimierung besteht in der Nutzung des Texturspeichers, der insbesondere für 2D-Zugriffsmuster - so wie es beim Rendern von Bildern notwendig ist - optimiert

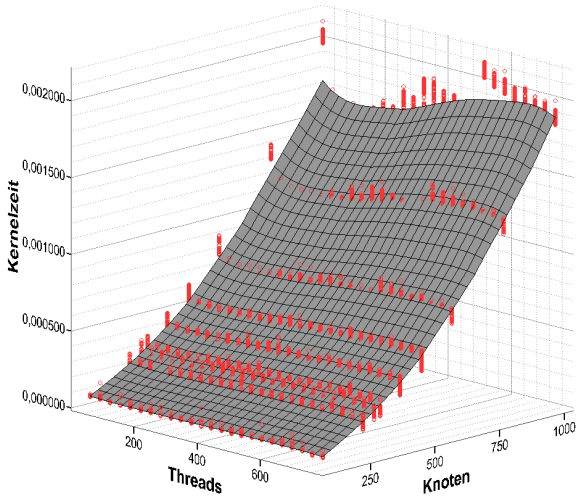


Abbildung 4.3: Kernelzeit des Relocate-Operators bei direkt berechneten Distanzen und Koordinaten im Texturspeicher

ist. Abbildung 4.3 zeigt in Abhängigkeit der Thread- und Knotenanzahl die Kernelzeit. Auch hier findet sich wieder der gleiche Verlauf der Kernelzeit, d.h., zunächst nimmt mit steigender Threadzahl die Kernelzeit ab, um dann wieder anzusteigen. Die Nutzung des Texturspeichers zur Ablage der Koordinaten ermöglicht jedoch keinen Performancegewinn. Die Kernelzeiten entsprechen in etwa denen, die die Distanzen bzw. die Koordinaten direkt aus dem globalen Speicher beziehen.

Bisher wird von der allgemeinen Auffassung ausgegangen, dass es vorteilhaft sein kann, mehr Berechnungen auf der GPU durchzuführen statt Speicherzugriffe zu verwenden. Bei der Verwendung einer CPU ist es hingegen in den meisten Fällen vorteilhaft, nicht jedes Mal, wenn eine Distanz benötigt wird, diese neu zu berechnen, sondern auf eine Distanzmatrix zurückzugreifen, die einmalig berechnet und danach im Arbeitsspeicher vorgehalten wird. Dieser Umstand soll hier näher im Kontext der GPU betrachtet werden. Dazu zeigt Abbildung 4.4 die Kernelzeit, wenn bei Verwendung von Distanzen auf eine Matrix zurückgegriffen wird, die im globalen Speicher abgelegt ist. Hier zeigt sich ein etwas anderes Bild als zuvor, d.h., mit zunehmender Threadzahl pro Block sinkt die benötigte Kernelzeit. Es scheint sich dabei die Ausnutzung der Cache-Strukturen weniger stark bemerkbar zu machen als bei der Nutzung von Koordinaten als Ausgangsbasis, da die Wahrscheinlichkeit, in unmittelbarer zeitlicher Nähe mehrmals die gleiche Distanz nachzufragen eher gering ist, im Vergleich zur mehrmaligen Abfrage der gleichen Koordinate.[93] Die Ergebnisse zeigen jedoch auch, dass die Verwendung einer Distanzmatrix im Globalspeicher zu keiner Verminderung der Ausführungsdauer, sondern vielmehr zu einer Steigerung der Kernelzeit geführt haben.

Wie weiter oben bereits erwähnt, ist der Texturspeicher gerade für Speicherzugriffe, die einem 2D-

[93] Angenommen es ist ein Problem mit 1000 Kunden gegeben, so ergibt sich bei Nutzung von Koordinaten als Ausgangsbasis, dass mit einer Wahrscheinlichkeit von $\frac{1}{1000}$ die gleiche Koordinate abgefragt wird. Demgegenüber beträgt die Wahrscheinlichkeit, dass die gleiche Distanz abgefragt wird ca. $\frac{1}{1000 \cdot 1000}$. Das zeigt, dass hierbei die Cache-Speicher sehr viel seltener zum Einsatz kommen.

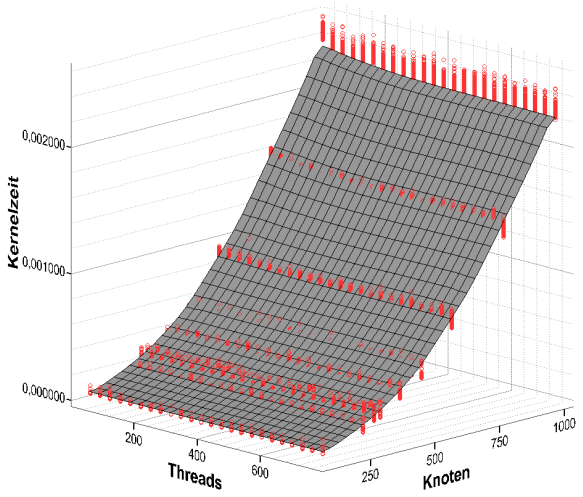


Abbildung 4.4: Kernelzeit des Relocate-Operators mit einer Distanzmatrix im globalen Speicher

Muster folgen, prädestiniert. Deshalb findet sich eine weitere Testreihe in Abbildung 4.5 bei der zum Speichern der Distanzmatrix der Texturspeicher zum Einsatz kommt. Hier zeigt sich ein ähnliches Bild wie zuvor unter Verwendung des Globalspeichers. Tendenziell bleibt die Kernelzeit unabhängig von der Threadanzahl konstant. Lediglich bei allzu starker Erhöhung der Threadanzahl steigt auch die Kernelzeit wieder an. Dieses Phänomen könnte den nun wieder besser nutzbaren Cache-Strukturen geschuldet sein, da der Texturspeicher eben auf 2D-Zugriffe fokussiert ist. Außerdem führt die Speicherung der Distanzmatrix im Texturspeicher zu einer deutlichen Reduzierung der Kernelzeiten im Vergleich zu den anderen hier vorgestellten Verfahren. Das heißt, als Gestaltungsempfehlung kann man sagen, dass es sinnvoll erscheint, gerade Distanzmatrizen, die einem 2D-Zugriffsmuster folgen, im Texturspeicher abzulegen. Auch Luong et al. (2013, S. 181) kommen zu den gleichen Ergebnissen. Sie speichern im Kontext des TSP die Problemdaten ebenfalls im Texturspeicher und erzielen damit bessere Ergebnisse, als bei der Verwendung des Globalspeichers. Allerdings reduzieren sich diese Vorteile des Texturspeichers gegenüber dem Globalspeicher im Rahmen der Fermi-Architektur, da nun auch der globale Speicher gecached ist (Schulz, 2013, S. 20). In der Vorgängerarchitektur war der Globalspeicher noch nicht gecached.

Die Swap-, Or-Opt-, 2-Opt*- und 2-Opt-Operatoren zeigen grundsätzlich ein ähnliches Verhalten wie der hier beschriebene Relocate-Operator (vgl. Abbildungen A.1, A.2, A.3 und A.4 im Anhang). In allen Fällen sorgt die Speicherung der Distanzmatrix im Texturspeicher der GPU für die geringste Kernelzeit und somit für die Empfehlung zur Nutzung dieses Speichers zur Ablage der Distanzmatrix. Unterschiede ergeben sich lediglich bei der Anzahl der Threads, die bei Verwendung des Texturspeichers anfallen. Beim Swap-Operator entspricht der Verlauf dem des Relocate-Operators. Bei den drei anderen Operatoren sinkt die Kernelzeit zunächst mit steigender Threadanzahl, um dann wieder anzusteigen. Das lässt sich eventuell dadurch erklären, dass bei diesen Operatoren auf eine andere Art und Weise die Kapazitäten berechnet werden. Beim Or-Opt-Operator werden zwei Zugriffe auf das Kapazitätsarray benötigt, was beim Relocate, der ihm am nächsten kommt, nicht der Fall ist. Beim 2-Opt* und 2-Opt

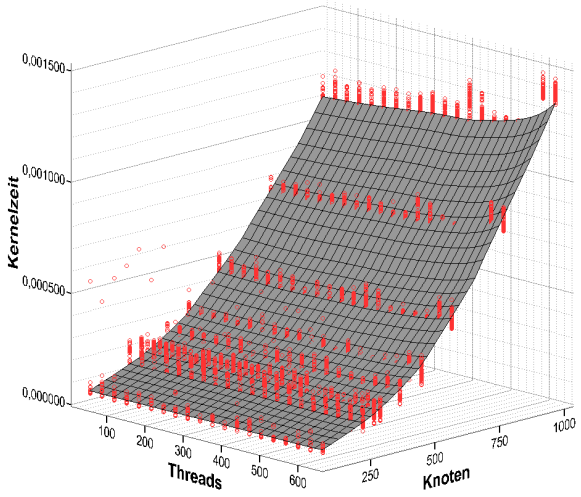


Abbildung 4.5: Kernelzeit des Relocate-Operators mit einer Distanzmatrix im Texturspeicher

sind schließlich zwei unterschiedliche Arrays beteiligt (Vorwärts- und Rückwärtskapazität), sodass es in diesem Fall wieder von Vorteil sein könnte, wenn Blöcke bereits zum Start vorbereitet werden, d.h. sich bereits im Idle-Modus auf der GPU befinden. Auch beim Cross-Exchange-Operator führt die Nutzung des Texturspeichers zu den besten Resultaten. Er wird jedoch im folgenden Abschnitt nochmals genauer betrachtet, da er durch seine flexiblen Konfigurationsmöglichkeiten eine Sonderstellung einnimmt.

Bevor er jedoch betrachtet wird, soll zunächst auf die Verteilung bzw. der Anteil der Kernelzeit an der gesamten GPU-Zeit im Rahmen des Relocate-Operators eingegangen werden. Die gesamte GPU-Zeit setzt sich zusammen aus der Kernelzeit zur Bewertung der Kanten, der Kernelzeit zur Bestimmung des besten Moves, sowie der Zeit, die zum Datentransfer (sowohl zur als auch von der GPU) benötigt wird. Hier zeigt sich auch, warum in der vorherigen Betrachtung die GPU-Zeit nicht im Gesamten herangezogen wird, sondern nur die Kernelzeit: die Anzahl der Threads, die pro Block gestartet wird, hat keinerlei Einfluss auf die Zeit, die zur Bestimmung des besten Moves benötigt wird, sowie auf die Transferzeit der Problemdaten (sowohl Distanzmatrix als auch bspw. Lösungsarray) von der CPU zur GPU und umgekehrt.

Die Kernelzeit zur Bestimmung des besten Moves ist unabhängig von dem verwendeten Operator, d.h., hier ist es nur notwendig, einmal die beste Konfiguration zu bestimmen. In Abbildung 4.6 ist die Kernelzeit zur Bestimmung des besten Moves in Abhängigkeit von der Anzahl der Threads und Knoten dargestellt. Man sieht wieder den bisher typischen Verlauf, dass einerseits mit steigender Knotenzahl die Laufzeit des Kernels ansteigt und andererseits bei konstant bleibender Knotenzahl mit steigender Threadanzahl die Kernelzeit sinkt bzw. mit weiter steigender Zahl wieder steigt. Die Anzahl der Threads, die hier getestet wird, entspricht der 2er-Potenz. Diese Einschränkung ergibt sich aus der Voraussetzung, die der Kernel zur Bestimmung des besten Moves verlangt. Das bedeutet, der Kernel liefert nur dann valide Ergebnisse, wenn die Anzahl der Threads einer 2er-Potenz entspricht. Weiterhin ist es bei der Bestimmung des besten Moves möglich, die maximale Anzahl von Threads pro Block (1024) auszuführen. In der Grafik kann man erkennen, dass die Kernelzeit mit steigender Anzahl der Threads sinkt, um

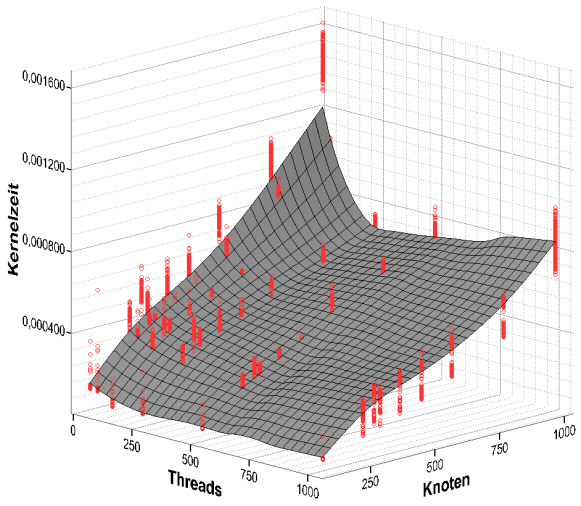


Abbildung 4.6: Kernelzeit zur Bestimmung des besten Moves

dann ab einer bestimmten Anzahl wieder anzusteigen. Insgesamt zeigt sich in den Versuchen, dass ein Kernel mit 512 Threads pro Block bei kleineren Instanzen marginal schneller arbeitet als einer mit 256 Threads. Letztere Konfiguration erzielt jedoch bei größeren Instanzen mit einer Knotenanzahl über etwa 450 bessere Ergebnisse. Aufgrund der besseren Performance der 256 Threads bei großen Instanzen, soll diese Konfiguration im Folgenden verwendet werden.

In Abbildung 4.7 ist das prozentuale Verhältnis von der Kernelzeit zur Bewertung des Moves, der Kernelzeit zur Bestimmung des besten Moves sowie der Transferzeit der Daten von der CPU zur GPU und umgekehrt dargestellt. Hierbei wird zwecks Bestimmung der Kernelzeit zur Bewertung der Moves eine Threadzahl von 128 selektiert, die nach den zuvor gezeigten Grafiken sehr gute Kernelzeiten erzielt hat. Es wird ersichtlich, dass gerade bei kleinen Instanzen die Transferzeit über die Hälfte der gesamten GPU-Zeit ausmacht. Das heißt, in Fällen mit kleinen Probleminstanzen spielt die Optimierung der Kernelzeit zur Bewertung der Moves nur eine untergeordnete Rolle. Wohingegen bei steigender Problemgröße der Anteil der Transferzeit immer weiter sinkt und der der Kernelzeit zur Bewertung der Moves weiter ansteigt, was eine intensive Optimierung des Kernels rechtfertigt. Der Anteil der Zeit zur Bestimmung des besten Moves liegt über alle betrachteten Problemgrößen hinweg auf einem relativ konstanten Niveau. Um diese Ergebnisse zu bestätigen, wird eine Korrelationsanalyse nach Pearson durchgeführt. In den Tabellen 4.2, 4.3 und 4.4 sind die Korrelationen der Knotenanzahl und verschiedenen Komponenten der GPU-Zeit dargestellt. Alle Faktoren steigen mit zunehmender Knotenanzahl an, da mehr Daten transferiert, mehr Daten bewertet und mehr Daten zur Bestimmung des besten Moves berücksichtigt werden müssen. Es kann jedoch bestätigt werden, dass die Kernelzeit zur Bewertung der Erzeugerkanten sehr viel stärker mit der Knotenanzahl korreliert und somit mit steigender Knotenanzahl immer größeren Einfluss auf die gesamte GPU-Zeit hat.

Demgegenüber steht die Transferzeit, die zwar auch mit der Knotenanzahl anwächst, allerdings sehr viel schwächer als beide Kernelzeiten. Das führt zu der Vermutung, dass das Versenden der Daten von der CPU zur GPU insbesondere dann „teuer“ ist, wenn der Bus zur Datenübertragung verwendet wird.

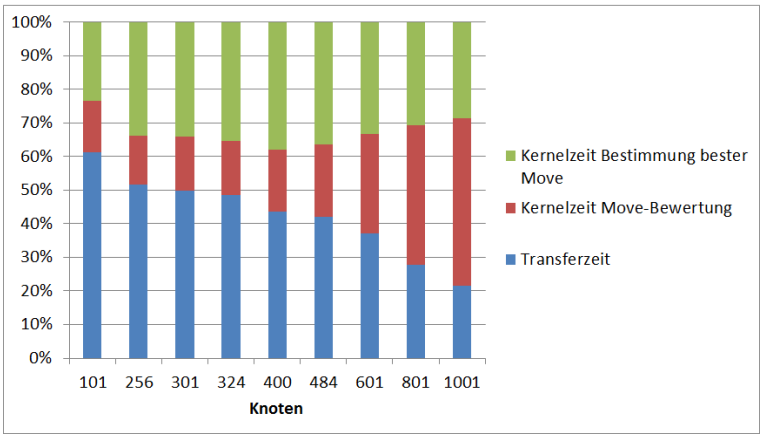


Abbildung 4.7: Zusammensetzung der GPU-Zeit (Relocate - 128 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,958**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,958**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0,01 (2-seitig)

Tabelle 4.2: Pearson-Korrelation zwischen der Kernelzeit zur Move-Bewertung und der Knotenanzahl

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,864**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,864**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0,01 (2-seitig)

Tabelle 4.3: Pearson-Korrelation zwischen der Kernelzeit zur Bestimmung des besten Moves und der Knotenanzahl

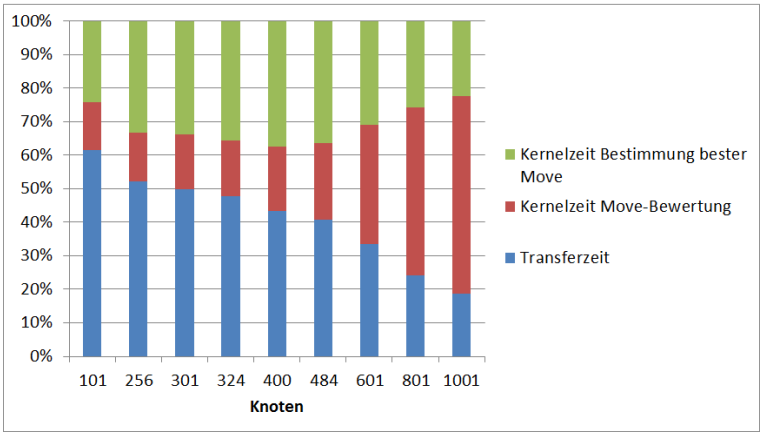


Abbildung 4.8: Zusammensetzung der GPU-Zeit (Swap - 128 Threads)

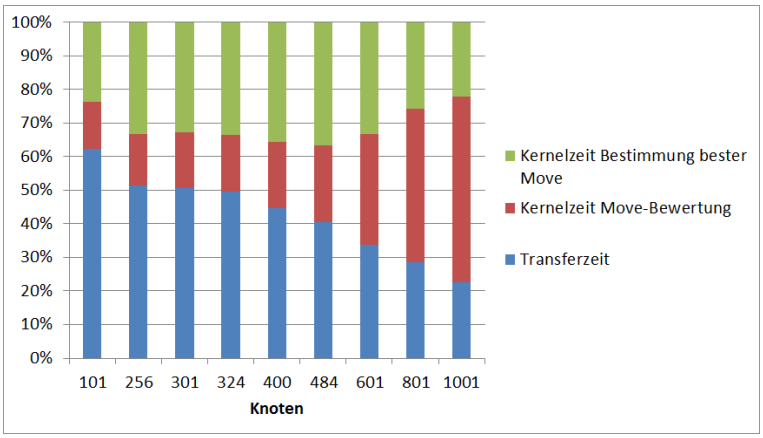


Abbildung 4.9: Zusammensetzung der GPU-Zeit (Or-Opt - 448 Threads)

Ist er einmal in Verwendung, spielt die Datenmenge eine weniger große Rolle. Insgesamt unterstützt das

		Knoten	Transferzeit
Knoten	Pearson-Korrelation	1	0,314**
	Sig. (2-seitig)		0,000
	N	900	900
Transferzeit	Pearson-Korrelation	0,314**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0,01 (2-seitig)

Tabelle 4.4: Pearson-Korrelation zwischen der Transferzeit und der Knotenanzahl

die Aussage von Abbildung 4.7, nach der eine Optimierung der Kernels zur Bewertung und Bestimmung des besten Moves von großer Bedeutung für den Erfolg bzw. die Performance eines Algorithmus sind.

Auch für den Swap-, Or-Opt-, 2-Opt*- und 2-Opt-Operator können die gleichen Beobachtungen gemacht werden wie aus den Abbildungen 4.8- 4.11 ersichtlich ist.

Die bisherigen Ausführungen zeigen, dass die Anzahl der Threads pro Block von fundamentaler Bedeutung für die Ausführungsdauer ist. Aufgrund dessen muss gerade dieser Parameter mit Sorgfalt gewählt werden. Grundsätzlich ergeben sich zwei Möglichkeiten zur Bestimmung dieses Parameters. Einerseits ist es möglich, einen statischen Threadplan aufzustellen, in dem genau determiniert wird, bei welcher Knoten- bzw. Kantenzahl welche Threadanzahl pro Block gestartet wird. Andererseits besteht die Möglichkeit, dass sich die Threadanzahl dynamisch ohne weitere Eingriffe von außen anpasst.

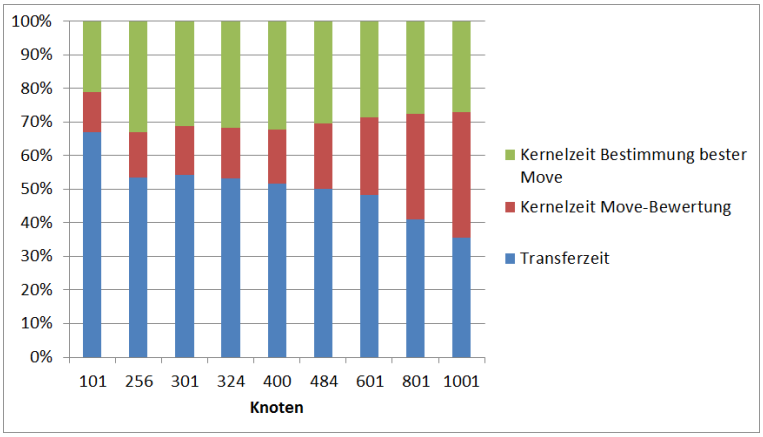


Abbildung 4.10: Zusammensetzung der GPU-Zeit (2-Opt* - 416 Threads)

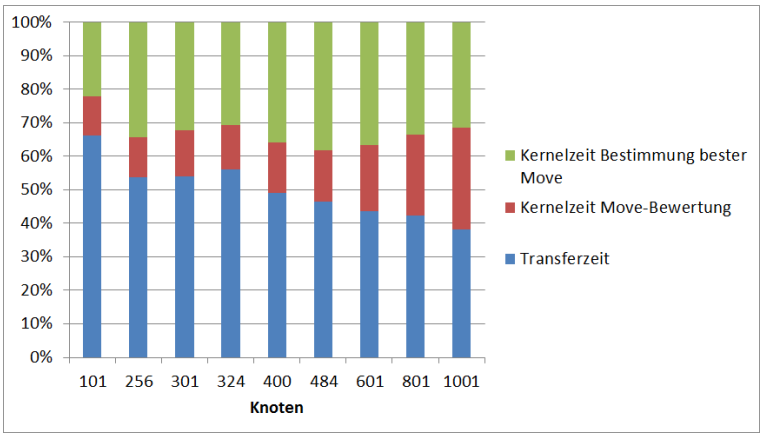


Abbildung 4.11: Zusammensetzung der GPU-Zeit (2-Opt - 256 Threads)

Erstgenannte Variante hat den Vorteil, dass direkt mit der passenden Anzahl von Threads gearbeitet werden kann. Sie hat aber den Nachteil, dass sehr viele unterschiedliche Probleminstanzen ex ante betrachtet, analysiert und bei jeder neu hinzugefügten Probleminstanz wieder neue Testreihen ausgeführt werden müssen.

Bei der dynamischen Anpassung hingegen ist es notwendig, dass die ersten Kernelaufrufe mit unterschiedlicher Threadzahl gestartet werden und nach einer gewissen Anzahl von „Testläufen“ bestimmt wird, mit welcher Konfiguration weiter gerechnet wird. Dieses Vorgehen bringt einerseits Flexibilität mit sich und andererseits eine im Gesamten etwas geringere Ausführungsgeschwindigkeit, da nicht von Beginn an mit der besten Threadanzahl gearbeitet wird. Trotz dieses Nachteils kommt in dieser Arbeit die dynamische Anpassung der Threadanzahl zum Einsatz. Wie man später noch sehen kann, kommen die lokalen Suchoperatoren im Rahmen mehrerer Metaheuristiken zum Einsatz und somit wird der Kernel jedes Operators im Verlauf mehrere tausend Male aufgerufen. Um die beste bzw. eine adäquate Threadanzahl zu bestimmen, wird der Kernel eines jeden Operators zu Beginn jeweils 10 Mal mit einer Threadanzahl von 32-768[94] (in 32er-Schritten) gestartet und die Zeit gemessen. Diese wird für jede Threadanzahl gemittelt und nachdem jede Konfiguration 10 Mal ausgeführt ist, die endgültige Konfiguration gewählt. Die endgültige Konfiguration ist die, mit der bis zum Ende des Verfahrens weiter gerechnet wird und welche den geringsten Mittelwert aufweist.[95] Auch wenn es sich nur um ein Näherungsverfahren handelt, ist die dadurch gewonnene Flexibilität nicht zu vernachlässigen. Dennoch sollten die vorherigen Ausführungen bzgl. der Thread- und Knotenanzahl einen Eindruck von der Wichtigkeit der Bestimmung der besten Threadanzahl sowie der Nutzung unterschiedlicher Speicherarten der GPU vermitteln.

4.1.1.2 Auswirkungen der Tabuliste auf die Kernelzeit

Nachdem im vergangenen Abschnitt auf die Besonderheiten bzw. die Auswirkungen, die aus unterschiedlicher Thread- und Knotenanzahl resultieren, eingegangen wurde, sollen in diesem Abschnitt die Besonderheiten bei Nutzung der Tabuliste beleuchtet werden. Bei Bestimmung des besten Moves hat die Tabuliste keine Auswirkung, da hier keine Prüfung stattfindet. Die Prüfung des Tabustatus findet lediglich im Kernel zur Bewertung der erzeugenden Kanten statt, weshalb dieser aus Sicht des Relocate-Operators näher betrachtet wird.

Abbildung 4.12 zeigt die Kernelzeit zur Bewertung der Erzeugerkanten in Abhängigkeit von der Anzahl der Threads sowie der Größe der Tabuliste. Dabei wird nur der Fall der Instanz mit 1001 Knoten betrachtet. In den Versuchen wird jede Threadkonfiguration von 32 - 512 (Vielfaches von 32) Threads durchlaufen. Für jede Threadanzahl werden pro Instanz zufällig 30 Fälle generiert, die in der Tabuliste zwischen 9 und 51 Einträge enthalten.[96] Weiterhin kommt die Version zum Einsatz, bei der die Distanzmatrix im Texturspeicher abgelegt wird, da diese in den vorherigen Auswertungen die besten

[94] Die 768 bilden das Maximum. Im Rahmen des Cross-Exchange-Operators oder bei Betrachtung einer Tabuliste ist es nicht mehr möglich, 768 Threads pro Block zu starten. Das heißt, in diesen Fällen wird die maximale Anzahl der zu testenden Threads nach unten korrigiert.

[95] Es ergeben sich pro Operator $10 \cdot \frac{768}{32} = 240$ Testdurchläufe, um die beste Threadanzahl zu bestimmen. Angenommen, es werden 100000 Kernelaufrufe ausgeführt, so ergibt das lediglich einen Anteil von $0,24\%$ an den gesamten Kernelaufrufen, der zur Bestimmung der besten Konfiguration aufgewendet werden muss.

[96] Durch die Tabuliste ist es mangels Ressourcen nicht möglich, bis zu 768 Threads wie in den bisher vorgestellten Versuchen zu starten, weshalb sich hier auf 512 Threads als Maximum beschränkt wird.

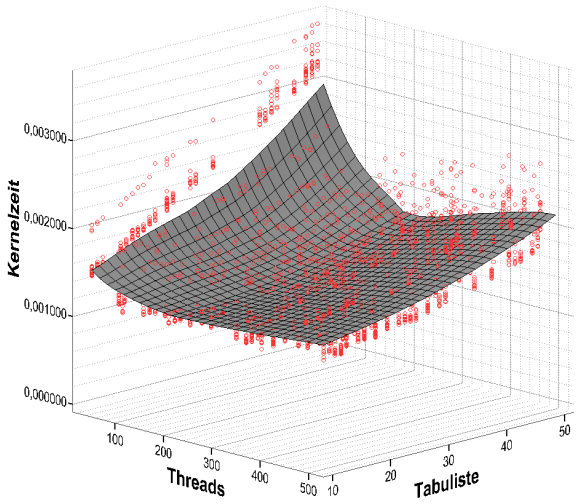


Abbildung 4.12: Einfluss der Tabuliste auf die Kernelzeit bei einer 1001-Knoten-Instanz

Resultate erzielt.

Hierbei zeigt sich einerseits, dass selbst bei einer geringen Länge der Tabuliste die Kernelzeit im Vergleich zu der Implementierung ohne Tabuliste ansteigt. Andererseits wird ersichtlich, dass mit zunehmender Länge der Tabuliste die Kernelzeit weiter ansteigt, was auch durch den Aufwand $O(TLL)$ zur Überprüfung des Tabustatus, wie in Kapitel 3.1.4 beschrieben, erklärt werden kann. Das heißt, das Laufzeitverhalten kommt dadurch zu Stande, dass bei Prüfung, ob ein Move tabu ist, alle Einträge in der Tabuliste durchlaufen werden müssen und somit die Dauer dieses Checks direkt die Gesamtlaufzeit beeinflusst. Findet eine adäquate Wahl der Threadanzahl statt, so ist es möglich, die Auswirkungen der Tabulistenlänge auf die Laufzeit zu verringern. In Abbildung 4.12 wird ersichtlich, dass gerade bei einer sehr geringen Anzahl von Threads die Laufzeit bei längerer Tabuliste sehr viel stärker ansteigt als dies bei einer höheren Threadanzahl der Fall ist.

Das Verhalten lässt sich sowohl für kleinere Probleminstanzen erkennen als auch für die anderen hier vorgestellten Operatoren, weshalb an dieser Stelle nicht weiter darauf eingegangen werden soll. Auch im Rahmen der Nutzung der Tabuliste findet in den weiteren Tests eine adaptive Anpassung der Threadanzahl pro Block statt.

4.1.1.3 Speedup der GPU-Implementierung im Vergleich zu einer CPU-Implementierung

Bisher wird lediglich der Frage nachgegangen, inwieweit die Konfiguration des Kernels gewählt werden muss, um eine möglichst geringe Ausführungsdauer zu erreichen. Vollkommen vernachlässigt wird jedoch das Verhältnis zur CPU. Hierbei soll weniger demonstriert werden, wie viel mehr Rechenleistung eine GPU im Vergleich zu einer CPU besitzt, sondern vielmehr sollen im Sinne der Argumentationskette von Schulz (2013, S. 21) die strukturellen Unterschiede, die sich durch die Verwendung der jeweiligen Plattform ergeben, aufgezeigt werden. Unter strukturellen Unterschieden wird verstanden, inwiefern sich der Speedup der GPU im Vergleich zur CPU ändert, wenn bspw. Probleme mit mehr Knoten betrachtet werden oder wie sich die Nutzung einer Tabuliste auf beiden Plattformen auswirkt. Anderson

et al. (2011) unterscheiden bei Publikationen, die Algorithmen auf der CPU und GPU testen, zwischen der Sicht des Anwendungsentwicklers und der Sicht des Computerarchitekten. Dabei fokussieren sich die Anwendungsentwickler auf die Möglichkeiten, die sich für die eigene Anwendung durch die Verwendung der GPU ergeben und vergleichen weniger die Architekturen der zugrundeliegenden Hardware (Anderson et al., 2011, S. 2). In dieser Arbeit soll der Leitlinie der Softwareentwickler gefolgt werden, da hier die verwendete Architektur nicht im Detail beschrieben wird, sondern es werden vielmehr die Auswirkungen der GPU auf Metaheuristiken im Bereich der Tourenplanung analysiert.

Bei den folgenden Vergleichen wird jeweils die GPU-Version, bei der die Distanzmatrix im Texturspeicher abgelegt wird, verwendet. Um den Speedup zu messen, sind die Zeiten, die zur Berechnung auf der CPU und GPU benötigt werden, notwendig. Dazu wird in dieser Arbeit bei der GPU-Zeit nicht nur von der reinen Kernelzeit ausgegangen wie dies in einigen Arbeiten der Fall ist (vgl. bspw. Che et al. (2008, S. 1373)), sondern es wird auch die Zeit berücksichtigt, die benötigt wird, um alle Daten von CPU zur GPU und umgekehrt zu transferieren. Hintergrund hierfür ist die Tatsache, dass ohne Daten auch keine Berechnungen auf der GPU durchgeführt werden können und der Datentransfer somit einen fundamentalen Bestandteil der Berechnungszeit auf der GPU darstellt. Man kann mit Gregg und Hazelwood (2011, S. 134) übereinstimmen und schlussfolgern, dass Vergleiche zwischen CPU und GPU erheblich verzerrt werden, wenn notwendige Speichertransfers nicht berücksichtigt werden. Dieser Berechnungsteil fällt nämlich bei reiner Betrachtung einer CPU-Version nicht an.[97] Die Verzerrungen, die daraus resultieren können, werden in der Arbeit von Gregg und Hazelwood (2011) mit umfangreichen Studien belegt.

Abbildung 4.13 zeigt den Speedup des Relocate-Operators der GPU-Version gegenüber der sequentiellen CPU-Version.[98] Wie zu erwarten, zeigt sich der Verlauf des Speedups in Analogie zum Verlauf der Kernelzeit, d.h., insbesondere bei kleiner Anzahl von Threads pro Block können über alle Problemklassen hinweg höhere Speedups erreicht werden. Gleichzeitig steigt der Speedup mit zunehmender Knotenanzahl unabhängig von der Zahl der gestarteten Threads an. Das bedeutet, dass die Zeit, die zur Bewertung der Erzeugerkanten und Berechnung des besten Moves auf der GPU benötigt wird zwar mit der Knotenanzahl ansteigt, wie aus Abbildung 4.5 ersichtlich ist, allerdings weniger stark als bei Verwendung der CPU. Auch Abbildung 4.14 unterstützt diesen Eindruck. Hierbei folgt das Wachstum der Ausführungsdauer der CPU-Version einer polynomiellen Regressionskurve, wohingegen die GPU-Version einer linearen Regressionsgeraden folgt. Der Speedup steigt mit wachsender Knotenanzahl, da die Ausführungsdauer ein unterschiedliches Wachstumsverhalten aufweist. Es sollte jedoch beachtet werden, dass nur eine Knotenanzahl bis 1001 in dieser Arbeit betrachtet wird. Es wird bei weiter steigender Knotenanzahl der Punkt erreicht werden, an dem auch die GPU-Version ein polynomielles Wachstum aufweist. Dieser Punkt wird im Vergleich zur CPU-Version später erreicht.[99] Weiterhin muss berücksichtigt werden, dass die GPU-Version ab einer bestimmten Knotenzahl in der hier implementierten Version an ihre Grenzen gelangen wird, da Ressourcen wie Register oder der Texturspeicher vollständig aufgebraucht sind und somit keine weiteren Kunden aufgenommen werden

[97] Außen vorgelassen wird hier lediglich der Datentransfer zur GPU, der anfällt, wenn die Problemdaten wie z.B. die Distanzmatrix übertragen werden. Da dies pro Aufruf eines gesamten Algorithmus nur einmalig stattfindet, kann diese Zeit unberücksichtigt bleiben.

[98] Das heißt, hier wird lediglich ein Kern der vier auf der CPU zur Verfügung stehenden verwendet.

[99] Bei noch weiter steigender Knotenanzahl (>>10000) werden die Anpassungskurven einem exponentiellen Wachstum folgen. Allerdings stellt sich in diesem Zusammenhang die Frage nach der praktischen Relevanz.

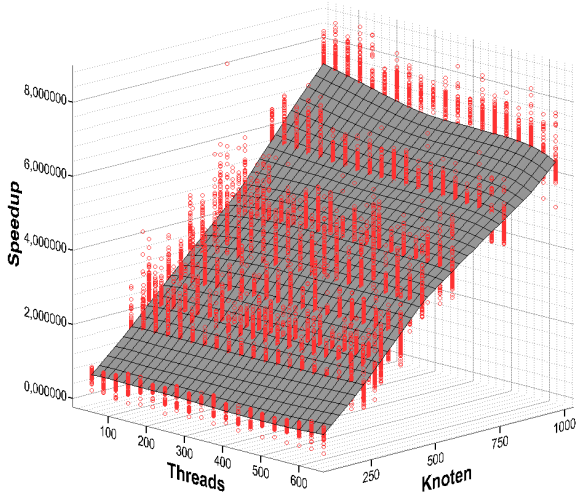


Abbildung 4.13: Speedup bei Anwendung des Relocate-Operators

können. Wenn dieser Fall in der CPU-Version auftritt, ist es relativ einfach möglich, den Arbeitsspeicher zu erhöhen und damit weiter zu arbeiten. Im Falle der GPU ist jedoch keine Erweiterungsmöglichkeit gegeben und man kann die maximal mögliche Speicherhöhe, die durch den Hersteller der Grafikkarte vorgegeben ist, nicht erweitern.[100]

Die Speedups des Swap-, Or-Opt-, 2-Opt*- und 2-Opt zeigen einen ähnlichen Verlauf wie der des Relocate-Operators (vgl. Abbildungen A.5 - A.8). Abgesehen vom Swap-Operator weisen alle ein etwa gleiches Speedup-Niveau im Maximum in der Höhe von ca. 7-8 aus. Der Swap-Operator nimmt in diesem Zusammenhang eine Sonderstellung ein, der einen maximalen Speedup von über 16 erreicht. Erklärt werden kann die Tatsache durch die Eigenschaft des Swap-Operators, dass er zur Klasse der 4-Opt-Moves zählt. Es müssen im Vergleich zu den anderen hier betrachteten Operatoren, die lediglich maximal 6 Distanzen benötigen, insgesamt acht Distanzen zur Bestimmung der Veränderung betrachtet werden. Hier kann der Fall eintreten, dass der Cache des GPU-Texturspeichers bessere Ergebnisse erzielt als der CPU-Cache, da ersterer, wie bereits geschrieben, für 2D-Zugriffsmuster ausgelegt ist. Je mehr Zugriffe nach diesem Schema also nötig sind, umso stärker könnte sich das im Speedup im Vergleich zur CPU bemerkbar machen.

Nun soll noch die Auswirkung der Tabuliste auf den Speedup untersucht werden. Auch hier wird wieder eine rein sequentielle Version auf der CPU verwendet. Abbildung 4.15(a) zeigt den Verlauf des Speedups beim Relocate-Operator bei Betrachtung der 1001-Knoten-Instanz. Es wird deutlich, dass der Speedup mit steigender Tabulistenlänge ebenfalls ansteigt. Weiterhin ist bei Nutzung der Tabuliste ein insgesamt höherer Speedup im Vergleich zur Verwendung des Relocate-Operators ohne Prüfung des Tabustatus erreichbar. In Darstellung 4.15(b) wird der Zusammenhang nochmals deutlicher. Wenn keine Tabuliste zum Einsatz kommt, lohnt sich die Verwendung der Grafikkarte zur Unterstützung der Berechnung bei Instanzen mit weniger als ca. 100 Knoten nicht, da eine Verlangsamung der

[100] Die Aussage gilt nur für die vorliegende Implementierung. Es werden keine Spezialfunktionen wie SSE verwendet. Dadurch ist es durch einfaches Aufrüsten des Rechners mit Arbeitsspeicher möglich, größere Instanzen zu lösen.

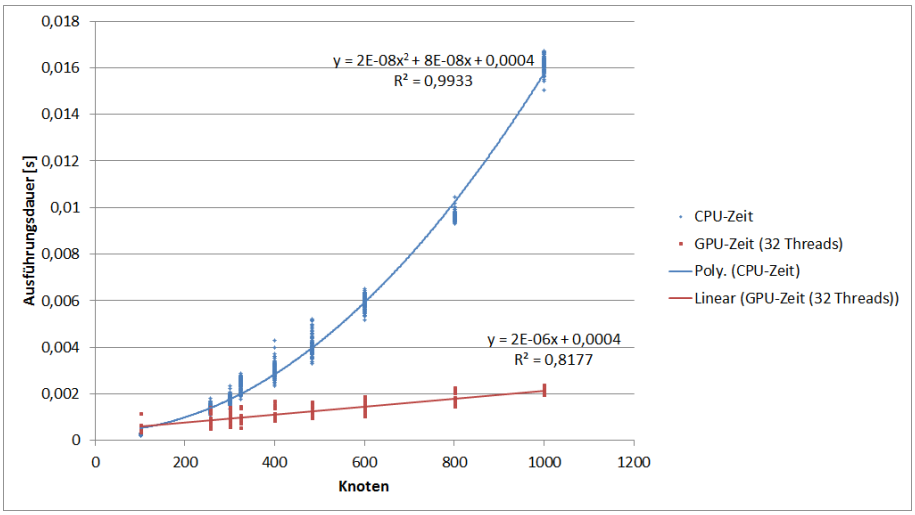


Abbildung 4.14: Regression über die Ausführungsdauer des Relocate-Operators in Abhängigkeit der Knotenanzahl (GPU-Version mit 32 Threads pro Block)

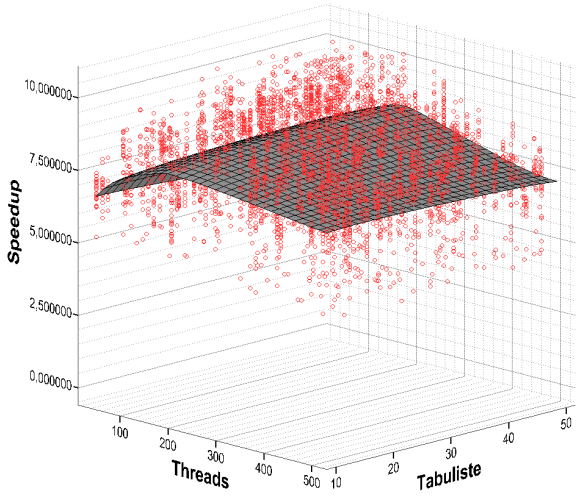
Ausführungsgeschwindigkeit im Verlgeich zur CPU stattfindet. Das ändert sich bei Verwendung einer Tabuliste. Ab einer bestimmten Größe der Tabuliste ist der Speedup größer als eins und somit ist unter Verwendung der GPU ein Geschwindigkeitsvorteil erzielbar. Weiterhin zeigt sich ein ähnliches Bild wie bei der Variation der Knotenzahl in Abbildung 4.13. Das bedeutet, die Kernelzeit steigt zwar ebenfalls mit der Größe der Tabuliste an, aber dieser Anstieg ist bei den betrachteten Größen weniger stark ausgeprägt als der Anstieg, wenn nur eine CPU verwendet wird. Auch für die anderen Operatoren ergibt sich ein vergleichbares Bild, weshalb auf diese nicht weiter eingegangen wird.

Zusammenfassend kann man feststellen, dass bei einer adäquaten Wahl der Threadanzahl pro Block der Nutzen - sprich der Speedup - der Parallelisierung stark von der Anzahl der Knoten sowie von der Größe der Tabuliste abhängig ist. Bei kleinen Instanzen mit weniger als ca. 100 Kunden sollte von der Parallelisierung abgesehen werden, da der Aufwand der Implementierung größer ist als ihr Nutzen. Dieses Verhältnis kehrt sich ab einer bestimmen Knotenzahl ins Gegenteil um, wodurch dann durch die Parallelisierung erhebliche Geschwindigkeitsvorteile erzielt werden können. Der Zeitpunkt bzw. die Anzahl der Knoten ab denen die Parallelisierung vorteilhaft ist, wird weiterhin von der Nutzung einer Tabuliste sowie deren Länge beeinflusst. So ist es durchaus auch bei kleinen Instanzen möglich, Geschwindigkeitsvorteile zu erzielen. Weiterhin muss beachtet werden, dass diese Gestaltungsempfehlungen nur für die genannte Hardwarekonfiguration gelten. Sobald bspw. eine schnellere CPU eingesetzt wird, wird sich das Verhältnis ändern. Gleiches gilt bei einer veränderten Grafikkarte.

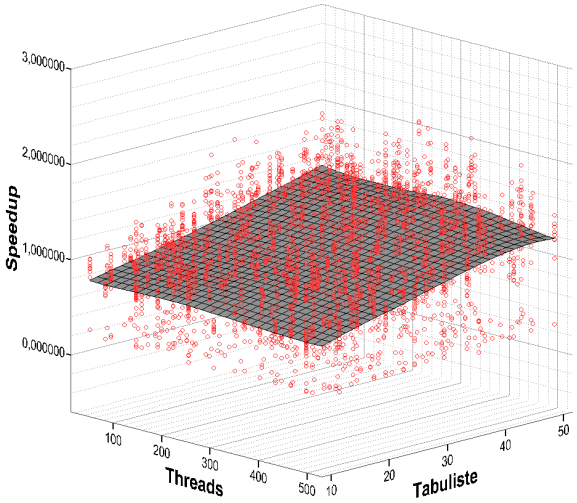
4.1.2 Cross-Exchange-Operator

Wie in Abschnitt 2.3.2.2 beschrieben besitzt der implementierte Cross-Exchange-Operator die Möglichkeit, die Anzahl der auszutauschenden Kunden zu variieren. Deshalb soll der Operator an dieser Stelle näher erläutert werden. Aufgrund der Menge an Kombinationsmöglichkeiten, mit denen der Cross-Exchange-Move eingestellt werden kann[101], soll sich auf vier Vertreter beschränkt werden. Dazu zählen der 1-1-, 3-3-, 1-5- sowie 5-5-Cross-Exchange. Die erstgenannten vertreten typische Fälle, wobei

[101] In dieser Arbeit werden insgesamt 25 Varianten verwendet.



(a) 1001 Knoten

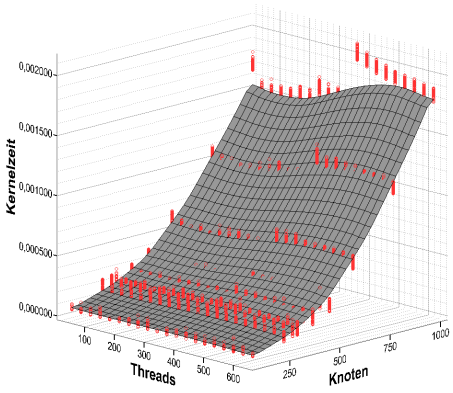


(b) 101 Knoten

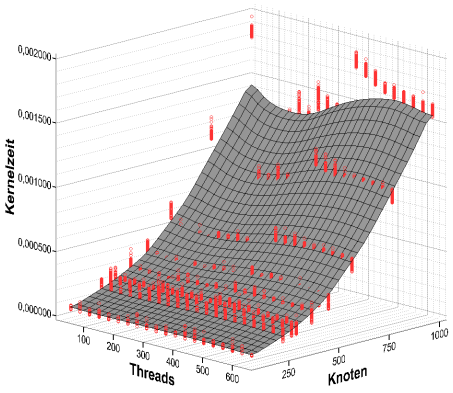
Abbildung 4.15: Speedup des Relocate-Operators bei Verwendung einer Tabuliste

der 1-1-Cross-Exchange nichts anderes als einen Swap-Move darstellt. Hier sollen insbesondere Vergleiche zu der direkten Implementierung des Swap-Moves gezogen werden. Die beiden letztgenannten stellen jeweils Extremfälle dar. Es wird an dieser Stelle nicht mehr auf die verschiedenen Möglichkeiten zur Ablage der Distanzmatrix eingegangen, da sich in diesem Zusammenhang das gleiche Bild wie bereits vorgestellt, zeigt, d.h., alle Darstellungen bzw. Ergebnisse beziehen sich auf den Umstand, dass die Distanzmatrix im Texturspeicher abgelegt ist.

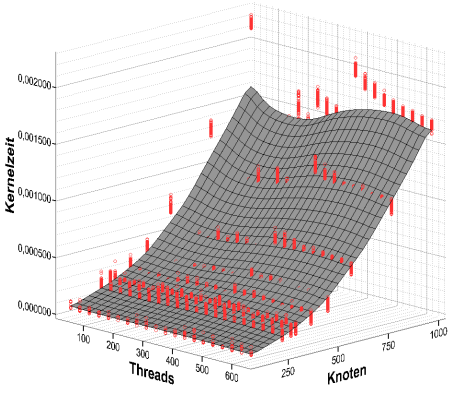
Abbildung 4.16 zeigt die Verläufe der Kernelzeiten in Abhängigkeit der Thread-, Knotenzahl sowie des Typs des verwendeten Cross-Exchange-Operators. Bei Betrachtung des 1-1-Cross-Exchange-Operators



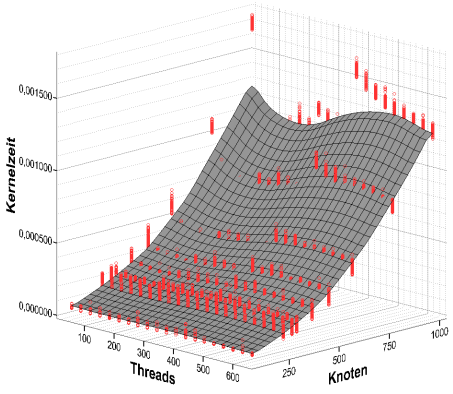
(a) 1-1-Cross-Exchange



(b) 3-3-Cross-Exchange



(c) 1-5-Cross-Exchange



(d) 5-5-Cross-Exchange

Abbildung 4.16: Kernelzeiten des Cross-Exchange-Operators in Abhängigkeit von der Thread- und Knotenanzahl

und seinem Gegenpart, dem Swap-Operator fällt auf, dass sich die besten erreichten Kernelzeiten nur geringfügig unterscheiden. Es lässt sich ein leichter Vorteil auf Seiten der Implementierung des 1-1-Cross-Exchanges feststellen, was dadurch begründet werden kann, dass dieser weniger Kanten bewerten muss, da er lediglich Inter-Route-Moves betrachtet, wohingegen die Swap-Implementierung auch Intra-Route-Moves bewertet. Das heißt, dass trotz einer im Vergleich zum Swap-Operator aufwendigeren Überprüfung der Kapazitätsrestriktion im 1-1-Cross-Exchange eine geringere Kernelzeit erreicht wird, da weniger Erzeugerkanten geprüft werden müssen.

Der Verlauf der Kernelzeiten ist bei allen Cross-Exchange-Operatoren, die in dieser Arbeit eingesetzt werden, gleich. Zunächst sinkt die Ausführungsdauer des Kernels mit steigender Threadzahl, um dann im weiteren Verlauf wieder anzusteigen.[102] Trotzdem finden sich z.T. erhebliche Unterschiede zwischen den Konfigurationen des Operators im Hinblick auf die Kernelzeit. So fällt auf, dass die 1-1-Konfiguration

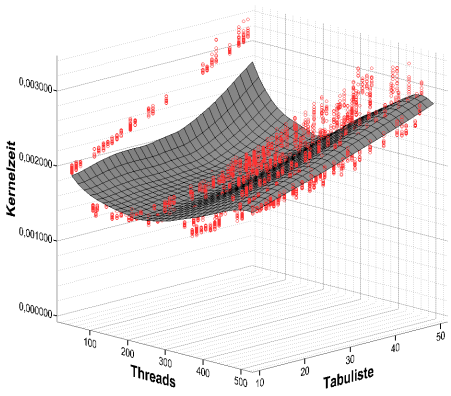
[102] Da die Kapazitätsberechnung des Cross-Exchange-Operators mehr Ressourcen benötigt als die im vorigen Abschnitt vorgestellten, ist es aufgrund der beschränkten Hardwareressourcen nicht möglich mehr als 640 Threads pro Block zu starten, weshalb sich in diesem Abschnitt auf Threads zwischen 31 und 641 pro Block beschränkt wird.

die höchste Kernelzeit besitzt. Eine mögliche Erklärung dafür resultiert aus der Prüfung von Kriterien wie sie in Abbildung 3.6 dargestellt sind. Wie bereits ausgeführt, gehört zu diesen Kriterien bspw., ob die Kundensequenzen, die ausgetauscht werden sollen, sich jeweils auf einer Route befinden und ob in der Sequenz kein Depot enthalten ist. Das führt zu dem Schluss, dass die Wahrscheinlichkeit, dass eine Sequenz von Kunden über mehrere Routen hinweg verläuft, mit steigender Länge höher ist. Wenn dem so ist, findet keine weitere Bewertung des Moves statt, sondern es wird direkt in das Ergebnisarray geschrieben. Dann findet weder die Distanzberechnung noch die Prüfung der Kapazität statt. Bei Betrachtung des 1-1-Cross-Exchange ist faktisch sicher, dass sich die Kundensequenz nur auf einer Route befindet, wohingegen das im Falle der anderen hier gezeigten Konfigurationen nicht unbedingt gegeben ist, weshalb dann weniger Kanten bewertet werden. Die Selektion sorgt also dafür, dass der höhere Aufwand, der mit der Kapazitätsberechnung einhergeht, wieder vermindert wird, da keine Kapazitätsprüfung vonnöten ist.

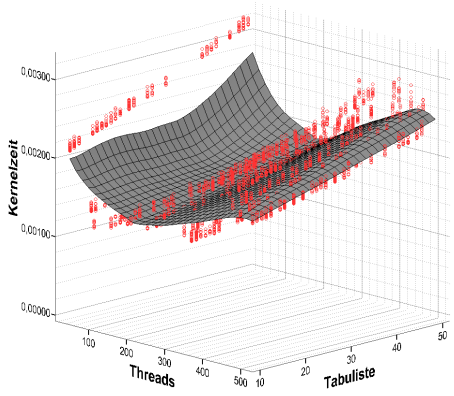
Auch der Verlauf der Kernelzeit bei Nutzung einer Tabuliste weist im Vergleich zu den vorherigen Ergebnissen keine Überraschungen auf. In Abbildung 4.17 sind die Zusammenhänge für die 1001-Knoten-Instanz dargestellt. Bei allen vier Varianten kann man den typischen Verlauf erkennen, dass die Kernelzeit unabhängig von der Größe der Tabuliste mit steigender Threadzahl sinkt, um danach bei weiterem Anstieg ebenfalls wieder anzusteigen. Wird die Anzahl der Threads festgesetzt, so wird auch hier der bereits bekannte Verlauf deutlich, dass mit steigender Länge der Tabuliste die Kernelzeit ebenfalls ansteigt.

Der Verlauf des Speedups bei Vernachlässigung einer Tabuliste ist in Abbildung A.9 visualisiert. Hierbei zeigt sich wieder der Verlauf, dass unabhängig von der Threadkonfiguration der Speedup mit wachsender Knotenanzahl ebenfalls ansteigt. Auch hier greift die Erklärung, dass die GPU-Zeiten weniger stark ansteigen als die CPU-Zeiten. Solange gewährleistet ist, dass genügend große Problemdaten vorhanden bzw. zu berechnen sind, ist es vorteilhaft, die Berechnungen auf die GPU auszulagern. Das ändert sich, wenn nur kleine Probleminstanzen bearbeitet werden. Auch hier muss wieder berücksichtigt werden, dass diese Aussagen bzw. Abbildungen jeweils nur für die hier betrachtete Hardwarearchitektur gelten. Die Schwelle, ab der die Parallelisierung lohnenswert ist, sollte hardwareindividuell abgeschätzt werden.

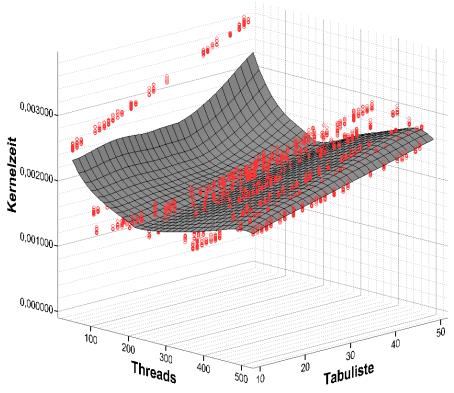
Ein etwas anderes Bild zeigt sich bei Betrachtung des Speedups unter Verwendung einer Tabuliste (vgl. Abbildung 4.18). Der Speedup des 1-1-Cross-Exchange steigt mit der Größe der Tabuliste. Weiterhin erzielt er einen leicht höheren Speedup als die Implementierung des Swap-Operators. Hierbei darf man jedoch nicht davon ausgehen, dass die Kernelzeit so viel schneller ist, sondern dass vielmehr die CPU-Version des 1-1-Cross-Exchange langsamer arbeitet als die des Swap-Operators und daraus die Speedupvorteile resultieren. Bei Betrachtung der anderen Speedupverläufe fällt auf, dass keine Geschwindigkeitsgewinne mehr durch die Erhöhung der Tabulistenlänge erzielt werden können. Gerade bei Betrachtung des 1-5-Cross-Exchange scheint sich der Speedup sogar mit zunehmender Steigerung der Tabulistengröße zu verringern. Das kann eventuell dadurch erklärt werden, dass durch Verwendung eines Cross-Exchange-Operators, der mehrere bzw. längere Kundensequenzen austauscht, die GPU - gerade bei der hier betrachteten 1001-Knoten-Instanz - an ihre Grenzen gelangt, d.h., dass die Laufzeiten durch Steigerung der Tabuliste stärker ansteigen als die CPU-Zeiten. Man kann vermuten, dass hier ein Plateau erreicht wird, ab dem keine weitere Steigerung des Speedups mehr möglich ist.



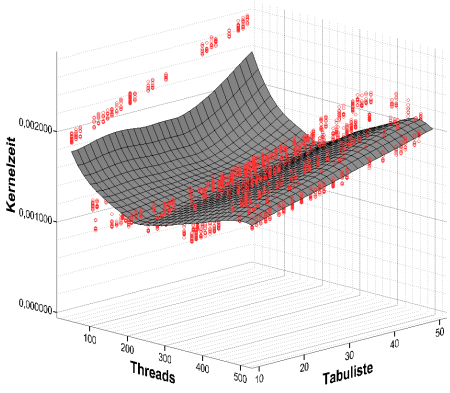
(a) 1-1-Cross-Exchange



(b) 3-3-Cross-Exchange



(c) 1-5-Cross-Exchange



(d) 5-5-Cross-Exchange

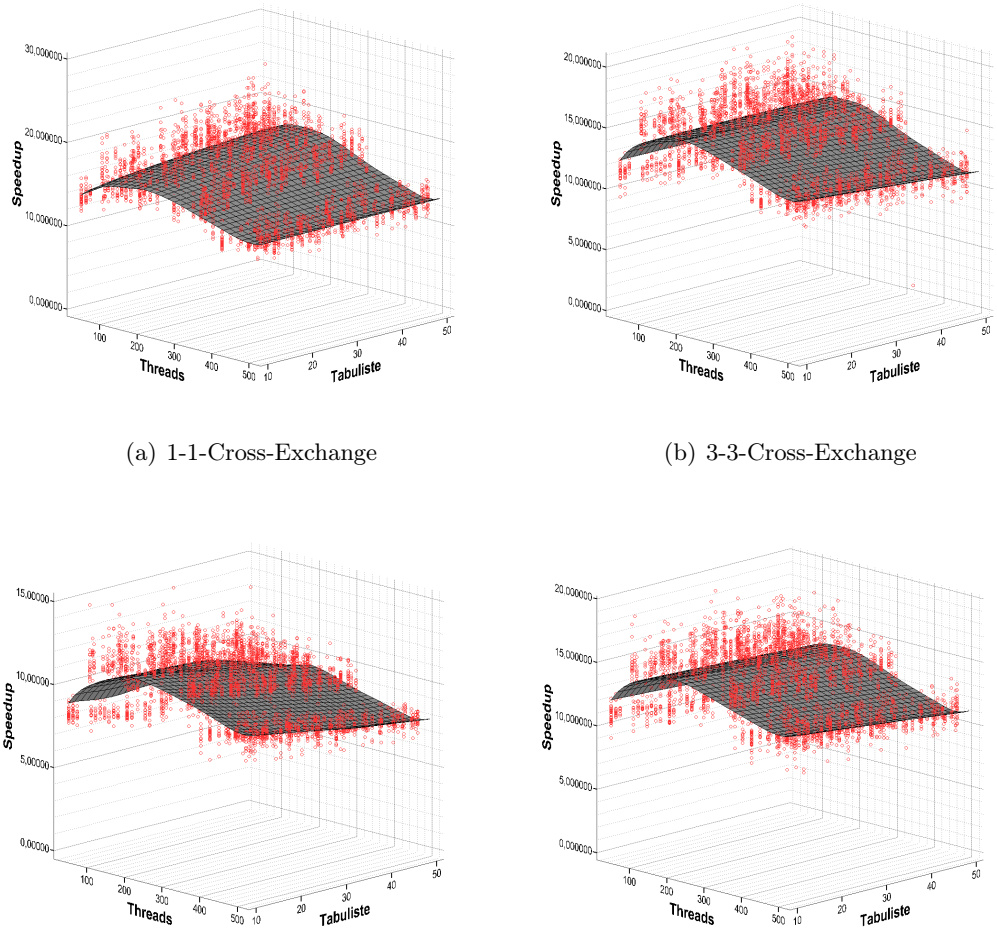
Abbildung 4.17: Kernelzeiten des Cross-Exchange-Operators in Abhängigkeit von der Threadanzahl und der Größe der Tabuliste (1001-Knoten-Instanz)

4.1.3 Weitere Kennzahlen zur Charakterisierung der lokalen Suchoperatoren

Auf Seite 202 findet sich eine Tabelle, die einige der Kennzahlen enthält, die in Kapitel 2.4.2 vorgestellt sind und zum Vergleich eines sequentiellen mit einem parallelen Algorithmus dienen. Hierbei wird für jeden in dieser Arbeit enthaltenen Operator die beste Threadkonfiguration für jede Testinstanz ermittelt. Die beste Threadkonfiguration ist dabei definiert als die Konfiguration, die den kleinsten Mittelwert über die GPU-Zeit der jeweiligen Instanz aufweist (entspricht $T_p(n)$). Als Referenzwert für den sequentiellen Algorithmus dient ebenfalls der Mittelwert über die CPU-Zeit der jeweiligen Instanz. Das heißt, der Mittelwert wird gleich $T^*(n)$ gesetzt.

Betrachtet werden sollen an dieser Stelle lediglich die Operatoren, die keine Tabuliste bei der Movebewertung berücksichtigen. Bei Verwendung der Tabuliste, kommt es jedoch zu ähnlichen Ergebnissen.

Die größten Speedups werden im Zusammenhang mit dem Cross-Exchange-Operator erreicht, und wenn die maximale Kundenanzahl von 1001 betrachtet wird. Neben dem Speedup, auf den nicht mehr weiter eingegangen werden soll, da er bereits weiter oben ausführlich behandelt wird, werden in der



(a) 1-1-Cross-Exchange

(b) 3-3-Cross-Exchange

(c) 1-5-Cross-Exchange

(d) 5-5-Cross-Exchange

Abbildung 4.18: Speedup des Cross-Exchange-Operators in Abhängigkeit von der Threadanzahl und der Größe der Tabuliste (1001-Knoten-Instanz)

Tabelle die Kosten, der Overhead sowie die Effizienz ermittelt.

Zur Bestimmung der Kosten ist es notwendig, dass die Zeit, die die parallele Version des Algorithmus zum Ausführen benötigt, mit der Anzahl der zur Verfügung stehenden Prozessoren multipliziert wird (vgl. Gleichung 2.35). Hieraus resultiert die Problematik, dass keine direkte Vergleichbarkeit zwischen den Prozessoren bzw. Kernen der GPU und denen der CPU gegeben ist. Das heißt, theoretisch werden die Aufgaben von der GPU mittels 512 leichtgewichtigen Prozessoren erledigt, die jeder für sich keineswegs mit einem Kern eines Intel i7 vergleichbar sind. Dennoch soll angenommen werden, dass 512 Prozessoren eingesetzt werden. Um sich jedoch der verwendeten CPU anzugleichen, scheint es sinnvoll, nicht nur mit den „kleinen“ Prozessoren der GPU zu arbeiten, sondern vielmehr mehrere dieser Prozessoren zusammenzufassen. Hierfür bietet sich die zugrundeliegende Hardwarearchitektur an. Wie in Kapitel 2.7.3 beschrieben, beherbergt ein SM der GPU 32 SPs. Somit bietet es sich im Falle der Nvidia GTX 580 an, als Anzahl der Prozessoren $p = 16$ anzunehmen. Diese Zahl entspricht genau der Anzahl der SMs. Hier wird deutlich, wie problematisch Vergleiche zwischen einer GPU und CPU sind. Dem Autor sind keine Ansätze bekannt, die sich dieser Problematik annehmen, weshalb in dieser Arbeit

mittels der beschriebenen Annahmen gearbeitet wird.

Aus den Kosten lässt sich ableiten, ob ein Algorithmus kostenoptimal arbeitet. Um Kostenoptimalität zu erreichen, ist es erforderlich, dass die Ausführungsdauer des parallelen Verfahrens der des sequentiellen entspricht. Bei der Annahme, dass 512 Prozessoren auf der GPU zum Einsatz kommen, wird keine Kostenoptimalität erreicht, wohingegen bei der Annahme, dass nur 16 Prozessoren verwendet werden, einige Operatoren die Kostenoptimalität erreichen. Das sind in diesem Fall genau die Operatoren, die einen Speedup von 16 oder mehr erreichen.[103]

In eine ähnliche Richtung wie die Kostenoptimalität gehen die Effizienz und der Overhead. Es zeigt sich, dass bis auf wenige Ausnahmen stets ein Overhead vorhanden ist. Die Ausnahmen bilden die Operatoren, die auch die Kostenoptimalität erreichen, wenn man annimmt, dass 16 Prozessoren eingesetzt werden. Geht man von 512 Prozessoren aus, so liegt immer ein Overhead, der zum Großteil durch die Kommunikation zwischen GPU-CPU erklärt werden kann, vor. Bei Betrachtung der Effizienz zeigt sich das gleiche Bild, d.h., die meisten Operatoren - unabhängig, ob 16 oder 512 Prozessoren angesetzt werden - weisen eine Effizienz kleiner eins auf, wodurch sie als „suboptimal bezüglich [ihrer] Kosten“ (Bengel et al., 2008, S. 317) zu bewerten sind.

Zusammenfassend zeigt sich, dass die Kennzahlen nur mit Vorsicht zur Beurteilung eines parallelen Algorithmus auf der Grafikkarte genutzt werden können, da nicht einzuordnen ist, welche Anzahl der Prozessoren der GPU einem Kern der CPU entspricht. Als Maß für die Güte eines parallelen Programms scheint sich hierbei die Nutzung des Speedups förmlich aufzudrängen, da bei dieser Kennzahl keine Annahmen getroffen werden müssen. Deshalb werden hier die Gesetze aus Abschnitt 2.4.2.5 nicht weiter betrachtet.

4.2 Numerische Analysen der Metaheuristiken

Nachdem im vorangegangenen Abschnitt auf die Operatoren näher eingegangen wurde, soll nun betrachtet werden, inwiefern diese in eine Metaheuristik integriert werden können bzw. welche Resultate dabei erzielt werden. Als Untersuchungsobjekte kommen die Instanzen von Golden et al. (1998), Gehring und Homberger (1999), Christofides et al. (1979) und Taillard (1993) zum Einsatz, wobei jeweils nur Probleme mit Kapazitätsbeschränkungen verwendet werden. Weiterhin liegt der Fokus auf den Probleminstanzen von Golden et al. (1998) mit denen die Metaheuristiken getestet werden. Mittels der anderen Instanzen soll gezeigt werden, wie anpassungsfähig die Algorithmen sind und welche Lösungsgüte sie bei Betrachtung unterschiedlicher Probleminstanzen erzielen können.

Hierbei haben alle vorgestellten Metaheuristiken das gleiche Abbruchkriterium. Ziel soll es sein herauszufinden, wie sich die Algorithmen im Zeitverlauf verhalten. Dazu wird je nach Instanzgröße eine festgelegte Abbruchzeit, die aus Tabelle 4.5 entnommen werden kann, verwendet. Weiterhin wird jede Probleminstanz von jedem Algorithmus fünf Mal bearbeitet, da alle Algorithmen nicht deterministisch sind und somit ein Mittelwert gebildet werden kann und die Aussagekraft der Ergebnisse mit steigendem

[103] Hier zeigt sich die Problematik der Schätzung, dass die GPU mit 16 Prozessoren arbeitet. Es werden hierdurch superlineare Speedups, d.h. größer 16 erzielt, was bei dem Kommunikationsaufwand, der bei GPUs anfällt eher unwahrscheinlich sein sollte.

Stichprobenumfang steigt.[104]

Knotenanzahl	Zeit [s]
≤ 76	30
≤ 101	120
≤ 200	600
> 200	3600

Tabelle 4.5: Abbruchkriterien der implementierten Metaheuristiken

Im Folgenden werden zunächst die beiden Metaheuristiken $VNS \times SimA_{CPU}$ und TS_{CPU} analysiert, die beide noch einen Großteil der Berechnungen auf der CPU ausführen. Diese verwenden die Operatoren direkt ohne weitere Anpassungen. Danach werden die Metaheuristiken, die einen Großteil der Berechnungen direkt auf der GPU ausführen, betrachtet.

4.2.1 Analyse GPU-basierter Metaheuristiken mit hohem CPU-Anteil

Zunächst sollen die $VNS \times SimA_{CPU}$ und die TS_{CPU}, die beide viele Berechnungen auf der CPU ausführen, hinsichtlich ihrer Lösungsgüte und Laufzeit betrachtet werden.

4.2.1.1 Betrachtung der $VNS \times SimA_{CPU}$

Im Falle der $VNS \times SimA_{CPU}$ sind bei den Operatoren keine Anpassungen notwendig. Der Temperaturschedule, der zum Einsatz kommt, ist statisch. Dabei wird die Temperatur mit 100 initialisiert und nach jeder Anwendung eines Moves eine Temperaturanpassung vorgenommen. Die Abkühlungsrate beträgt $0,9999$. Eine mögliche Abbruchbedingung beim SimA ist das Erreichen der Temperatur nahe 0. In dieser Arbeit soll der Algorithmus eine Stunde laufen. Da bei einem Temperaturniveau nahe 0 fast keine schlechteren Lösungen mehr akzeptiert werden, sind andere Lösungen als die aktuelle nahezu unmöglich und der Rechner würde arbeiten, ohne bessere Resultate zu erzielen. Deshalb wird bei Erreichen einer Temperatur von $0,2$ diese wieder auf 5 angehoben. Diese Parameter haben sich in vorangegangen Tests als sehr wirksam erwiesen. Varanelli und Cohoon (1999, S. 485) bemerken dazu, dass Temperaturpläne, die ein Aufwärmen erlauben, zu teilweise besseren Ergebnissen führen können als monoton fallende Temperaturen. Weiterhin hat sich eine Temperatur von 1 im Zusammenhang mit dem Shaking-Verfahren als adäquater Parameter herauskristallisiert. Hierbei soll nochmals auf die klare Trennung der beiden Temperaturen hingewiesen werden. Während des Shakings wird die Akzeptanzwahrscheinlichkeit immer anhand der Temperatur von 1 ermittelt, unabhängig davon welches Temperaturniveau zur Berechnung der Akzeptanzwahrscheinlichkeit der anderen Moves verwendet wird.

In Tabelle 4.6 sind die Ergebnisse im Rahmen der Golden-Instanzen dargestellt. Wie bereits ausgeführt, beträgt das Abbruchkriterium des Algorithmus 3600 Sekunden, weshalb in der Tabelle lediglich die Zeit angegeben ist, wann die beste Lösung in allen fünf Durchläufen gefunden wurde. Weiterhin sind nochmals die besten bekannten Lösungen enthalten, sowie die durchschnittliche Abweichung von den

[104] Der relative kleine Stichprobenumfang resultiert aus der Rechendauer, die zur Lösung aller Instanzen benötigt wird. Um bspw. einen Stichprobenumfang von 10 zu erreichen, wären weitere 3 Monate Rechenzeit benötigt worden, was in keiner Relation zu den erzielbaren Ergebnissen gestanden hätte, d.h., die Ergebnisse mit einem Stichprobenumfang von fünf sollten sehr gute Einblicke in die Heuristiken ermöglichen.

besten Lösungen und die Abweichung der besten gefundenen Lösung von der besten bekannten Lösung. Des Weiteren enthält die Tabelle die besten Lösungen, die jeweils nach zwei, zehn und 30 Minuten gefunden wurden, sowie deren Abweichungen zur besten bekannten Lösung.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	584,88	583,97	3561,41	0,89	0,73
g10	736,26	742,99	741,68	1531,12	0,91	0,74
g11	912,84	921,33	920,77	3586,47	0,93	0,87
g12	1102,69	1113,28	1110,58	3093,95	0,96	0,72
g13	857,19	860,57	859,59	1449,65	0,39	0,28
g14	1080,55	1081,05	1080,55	2265,23	0,05	0,00
g15	1337,92	1350,88	1347,58	3385,47	0,97	0,72
g16	1612,50	1628,51	1624,22	3405,27	0,99	0,73
g17	707,76	708,03	707,80	2951,54	0,04	0,01
g18	995,13	1001,50	1001,14	3031,76	0,64	0,60
g19	1365,60	1372,73	1369,15	3252,78	0,52	0,26
g20	1818,25	1838,48	1836,42	2525,25	1,11	1,00
Summe	13106,40	13204,22	13183,46	34039,89	0,75	0,59

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	588,33	1,49	586,42	1,16	585,10	0,93
g10	745,28	1,23	742,93	0,91	741,68	0,74
g11	923,63	1,18	921,69	0,97	920,85	0,88
g12	1116,32	1,24	1111,61	0,81	1111,61	0,81
g13	865,48	0,97	860,95	0,44	859,59	0,28
g14	1084,92	0,40	1081,31	0,07	1080,89	0,03
g15	1355,84	1,34	1350,72	0,96	1350,72	0,96
g16	1641,62	1,81	1630,60	1,12	1629,72	1,07
g17	710,48	0,38	708,00	0,03	707,83	0,01
g18	1012,02	1,70	1004,16	0,91	1001,62	0,65
g19	1381,40	1,16	1373,71	0,59	1372,18	0,48
g20	1856,06	2,08	1841,85	1,30	1838,75	1,13
Summe	13281,37	1,33	13213,94	0,82	13200,54	0,72

Tabelle 4.6: Lösungen der $VNS \times SimA_{CPU}$ auf den Golden-Instanzen

Aus der Ergebnistabelle wird deutlich, dass das Verfahren auf den Golden-Instanzen sehr gute Ergebnisse erzielt. Die Abweichung der besten gefundenen Lösungen zu den besten bekannten beträgt im Mittel lediglich $0,59\%$. Auch die durchschnittliche Abweichung ist mit $0,75\%$ äußerst gering und beweist, dass das Verfahren sehr robust aufgestellt ist und die besten gefundenen Resultate keine Ausnahme darstellen. Im Falle der Instanz g14 kann die beste bekannte Lösung bestätigt werden. Bereits nach zwei Minuten Laufzeit liegt nur noch eine Abweichung von $1,33\%$ vor. Nach 10 Minuten ist der Abstand zur besten bekannten Lösung bereits auf unter 1% gesunken. Das verdeutlicht, dass das Verfahren auch dann eingesetzt werden kann, wenn wenig Zeit zur Berechnung zur Verfügung steht. Hierbei muss der Anwender selbst abwägen, ob ihm eine kürzere Laufzeit des Algorithmus oder ein besserer Zielfunktionswert wichtig ist bzw. ob überhaupt die Zeit für eine längere Berechnung vorhanden ist.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,67	524,61	18,88	0,01	0,00
c2	835,26	837,49	835,32	25,29	0,27	0,01
c3	826,14	827,93	826,21	14,26	0,22	0,01
c4	1028,42	1033,21	1029,87	454,88	0,47	0,14
c5	1291,29	1308,67	1303,94	398,66	1,35	0,98
c11	1042,11	1042,18	1042,11	335,59	0,01	0,00
c12	819,56	819,56	819,56	5,48	0,00	0,00
Summe	6367,39	6393,72	6381,62	1253,03	0,41	0,22

Tabelle 4.7: Lösungen der $VNS \times SimA_{CPU}$ auf den Christofides-Instanzen

In den Tabellen 4.7 und 4.8 finden sich die erzielten Lösungen der $VNS \times SimA_{CPU}$ auf den

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1621,24	1618,36	14,31	0,18	0,00
T75b	1344,62	1345,19	1344,64	24,39	0,04	0,00
T75c	1291,01	1291,87	1291,01	18,58	0,07	0,00
T75d	1365,42	1365,42	1365,42	10,26	0,00	0,00
T100a	2041,34	2060,57	2056,46	61,96	0,94	0,74
T100b	1939,90	1942,40	1941,38	108,97	0,13	0,08
T100c	1406,20	1411,24	1406,26	38,08	0,36	0,00
T100d	1580,46	1591,36	1585,07	18,46	0,69	0,29
T150a	3055,23	3060,24	3057,21	273,45	0,16	0,06
T150b	2727,20	2735,91	2732,12	512,16	0,32	0,18
T150c	2341,84	2362,50	2361,62	377,69	0,88	0,84
T150d	2645,40	2667,95	2662,82	499,06	0,85	0,66
T385	24366,69	24752,01	24674,85	2588,30	1,58	1,26
Summe	47723,67	48207,92	48097,22	4545,67	1,01	0,78

Tabelle 4.8: Lösungen der $VNS \times SimA_{CPU}$ auf den Taillard-Instanzen

Benchmarkinstanzen von Christofides et al. (1979) und Taillard (1993). Auch hier zeigt sich die Stärke des Verfahrens. So ist die mittlere Abweichung der besten gefundenen Lösungen nur $0,22\%$ bzw. $0,78\%$. Der höhere Wert bei den Taillard-Instanzen resultiert vorwiegend aus Instanz T385, bei der die Abweichung über 1% liegt. Insgesamt erreicht das Verfahren bei den Christofides-Instanzen annähernd 5 Mal die beste bekannte Lösung und bei den Taillard-Instanzen ebenfalls 5 Mal. Diese Ergebnisse unterstreichen, dass das Verfahren nicht nur für stark strukturierte Probleme, wie das bei den Golden-Instanzen der Fall ist, geeignet ist, sondern ebenso für andere Probleminstanzen.

In Tabelle 4.9 sind die Ergebnisse, die auf den Gehring-Instanzen erzielt werden, dargestellt. Hierbei werden weiterhin die C2-Instanzen mitaufgeführt - auch wenn die Resultate von Mester und Bräysy (2007) nicht stimmen können. Bei Berechnung der durchschnittlichen Abweichungen werden diese Instanzen jedoch nicht berücksichtigt. Gleiches gilt für die im weiteren Verlauf der Arbeit vorgestellten Metaheuristiken. Die Gesamtresultate bzw. die Gesamtabweichung der besten gefundenen Lösung im Kontext der Instanzen von Gehring und Homberger (1999) beträgt ca. 0,5% (vgl. Tabelle 4.9). Allerdings sind hier die C1-Instanzen sowie die Instanz RC1_400 hervorzuheben, für die die $VNS \times SimA_{CPU}$ neue beste Lösungen ermittelt hat.[105] Die größte Verbesserung mit über 1% wird auf Problem C1_800 erzielt. Auch die anderen Ergebnisse sind sehr vielversprechend. Im Rahmen der C2-Instanzen sieht man die erheblichen Abweichungen, die die geringere Fahrzeuganzahl, die von Mester und Bräysy (2007) verwendet wird, mit sich bringt. Aber wie bereits beschrieben, sind die Ergebnisse in diesem Zusammenhang nicht vergleichbar und werden deshalb hier und im Folgenden ausgeklammert.

Alles in allem kann man schlussfolgern, dass die zuvor vorgestellten Operatoren mit einfachsten Mitteln in eine Metaheuristik integriert und dabei sehr gute Ergebnisse erzielt werden können. Um die Lösungsgüte zu erreichen, ist es nicht notwendig, aufwendige Operatoren oder Crossover-Verfahren zu implementieren; es genügen die in der Literatur eingängig beschriebenen lokalen Suchoperatoren. Weiterhin lässt sich der Algorithmus auf unterschiedlichste Problemklassen anwenden und er erreicht eine konstant hohe Lösungsgüte.

[105] Durch die Problematik der zu geringen Fahrzeuganzahl in den Resultaten von Mester und Bräysy (2007) werden im Rahmen des vorliegenden Beitrags auch neue beste Lösungen für die C2-Instanzen geliefert. Allerdings werden bei Aussagen zu neuen besten Lösungen im Text stets die C2-Instanzen ausgeklammert, da für diese bisher keine gültigen Ergebnisse veröffentlicht sind.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2568,17	**2567,68**	1571,77	-0,09	**-0,11**
C1_400	6765,03	6740,37	**6735,41**	3535,60	-0,36	**-0,44**
C1_600	13465,24	13472,17	**13462,80**	3479,26	0,05	**-0,02**
C1_800	23999,35	23729,69	**23690,76**	2636,06	-1,12	**-1,29**
C1_1000	39811,15	39757,91	**39699,29**	2284,45	-0,13	**-0,28**
C2_200	1368,89	1491,91	1484,34	2210,58	8,99	8,43
C2_400	2799,40	3142,38	3130,58	1779,31	12,25	11,83
C2_600	5441,98	6334,99	6323,97	3189,71	16,41	16,21
C2_800	8241,83	9742,12	9719,78	3086,06	18,20	17,93
C2_1000	11792,66	14547,99	14491,25	1684,73	23,36	22,88
R1_200	2878,24	2891,92	2887,31	1922,05	0,48	0,31
R1_400	7192,03	7265,90	7238,46	3345,59	1,03	0,65
R1_600	15691,37	15821,75	15700,29	3086,54	0,83	0,06
R1_800	27650,67	28082,02	27983,38	3393,58	1,56	1,20
R1_1000	42311,91	43124,13	42974,66	1441,62	1,92	1,57
R2_200	1610,30	1617,55	1611,24	1355,76	0,45	0,06
R2_400	3323,05	3349,89	3346,68	3161,34	0,81	0,71
R2_600	6254,17	6325,31	6295,97	3039,24	1,14	0,67
R2_800	10230,06	10328,97	10319,47	2381,46	0,97	0,87
R2_1000	15149,40	15220,91	15202,40	2873,72	0,47	0,35
RC1_200	2804,20	2824,32	2817,15	3586,45	0,72	0,46
RC1_400	7381,47	7351,44	**7331,79**	3531,75	-0,41	**-0,67**
RC1_600	14786,15	14972,17	14889,15	3101,35	1,26	0,70
RC1_800	26720,23	26925,36	26865,81	1082,27	0,77	0,54
RC1_1000	41583,54	42182,37	42023,63	3577,99	1,44	1,06
RC2_200	1513,00	1514,87	1513,00	1890,41	0,12	0,00
RC2_400	3100,06	3113,21	3111,64	534,65	0,42	0,37
RC2_600	5732,08	5835,96	5802,39	1902,57	1,81	1,23
RC2_800	9217,99	9359,65	9318,72	3562,44	1,54	1,09
RC2_1000	13631,76	13986,01	13879,33	114,40	2,60	1,82
Summe	375017,70	383621,42	382418,33	74342,73	0,87	0,55

Tabelle 4.9: Lösungen der $VNS \times SimA_{CPU}$ auf den Gehring-Instanzen

4.2.1.2 Betrachtung der TS_{CPU}

In Analogie zur $VNS \times SimA_{CPU}$ sind auch bei der TS_{CPU} keine großen Anpassungen hinsichtlich der Operatoren notwendig. Der einzige Unterschied besteht in der Verwendung einer Tabuliste. Mit der wichtigste Parameter bei der TS ist deren Länge. In der Implementierung wird zu Beginn eines Durchlaufs zufällig eine Länge zwischen 4 und 16 gewählt, da sich diese Größe in vorangegangenen Tests als sehr wirksam herausgestellt hat. Wie in Abschnitt 3.1.4 beschrieben, muss weiterhin eine untere bzw. obere Grenze für die Zielfunktionswerte, die tabu sind, gesetzt werden. Als Initialisierung wird eine untere Grenze von 0,0001 und eine obere Grenze von 0,0001 gewählt. Damit soll erreicht werden, dass annähernd nur die Lösung als solche den Tabustatus erhält. Weiterhin wird im Rahmen der TS nicht immer eine Diversifikation bzw. ein Shaking durchgeführt, sondern nur alle 600 Iterationen, wenn dazwischen keine neue beste Lösung gefunden wurde. Außerdem wird nach jeweils 10000 Iterationen, in denen keine neue beste Lösung im Suchverlauf ermittelt wurde, die bisherige beste gefundene Lösung wieder als Ausgangslösung gesetzt. Die obere und untere Grenze für die Tabuliste wird nach jedem Move aktualisiert. Wenn eine neue beste Lösung gefunden wird, wird die Grenze entsprechend der initialen Voreinstellung gesetzt, andernfalls wird jeweils zufällig eine Zahl zwischen 0,001 und 2,0 für die obere und untere Grenze gewählt. Zur Diversifizierung wird in Analogie zu $VNS \times SimA_{CPU}$ konstant eine Temperatur von 1 gewählt, um die Akzeptanzwahrscheinlichkeit des Shaking-Moves zu ermitteln.

In Tabelle 4.10 sind die Ergebnisse der TS für die Instanzen von Golden et al. (1998) dargestellt. Man kann erkennen, dass die Resultate sich nur marginal von denen der $VNS \times SimA_{CPU}$ unterscheiden. Im Vergleich der Lösungsgüte liegt die TS_{CPU} knapp vorne, wobei die Unterschiede zu gering sind, um daraus Empfehlungen für und wider eines der Verfahren aussprechen zu können. Auch bei Betrachtung geringerer Ausführungszeiten zeigen sich nur minimale Unterschiede zur vorgestellten VNS. Die Ergeb-

nisse unterstreichen jedoch, dass man mit relativ einfachen Mitteln bzw. Metaheuristiken sehr gute Lösungsgüten erzielen kann.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	582,84	582,35	583,45	0,54	0,45
g10	736,26	740,05	739,00	3582,69	0,52	0,37
g11	912,84	917,33	916,01	2765,09	0,49	0,35
g12	1102,69	1111,59	1110,50	3379,05	0,81	0,71
g13	857,19	861,32	859,55	2502,81	0,48	0,28
g14	1080,55	1081,23	1080,89	2385,03	0,06	0,03
g15	1337,92	1345,45	1344,51	1892,89	0,56	0,49
g16	1612,50	1622,07	1618,46	3045,72	0,59	0,37
g17	707,76	708,07	707,91	1702,58	0,04	0,02
g18	995,13	1000,33	997,96	2717,35	0,52	0,28
g19	1365,60	1380,20	1375,70	3205,09	1,07	0,74
g20	1818,25	1850,77	1849,45	3562,72	1,79	1,72
Summe	13106,40	13201,24	13182,29	31324,46	0,72	0,58

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	583,23	0,61	582,35	0,45	582,35	0,45
g10	743,84	1,03	742,70	0,87	739,74	0,47
g11	922,69	1,08	917,01	0,46	916,56	0,41
g12	1119,96	1,57	1112,47	0,89	1110,52	0,71
g13	864,78	0,89	862,86	0,66	860,84	0,43
g14	1085,47	0,46	1081,44	0,08	1080,89	0,03
g15	1349,31	0,85	1346,05	0,61	1344,68	0,50
g16	1629,67	1,07	1625,77	0,82	1621,16	0,54
g17	708,35	0,08	708,22	0,07	707,91	0,02
g18	1004,71	0,96	1001,50	0,64	999,48	0,44
g19	1394,09	2,09	1384,65	1,40	1378,08	0,91
g20	1872,20	2,97	1859,56	2,27	1851,57	1,83
Summe	13278,30	1,31	13224,58	0,90	13193,79	0,67

Tabelle 4.10: Lösungen der TS_{CPU} auf den Golden-Instanzen

In den Tabellen 4.11, 4.12 und 4.13 finden sich die Ergebnisse der TS_{CPU} für die Instanzen von Christofides et al. (1979), Taillard (1993) sowie Gehring und Homberger (1999). Hierbei kann man deutliche Unterschiede zur zuvor vorgestellten $VNS \times SimA_{CPU}$ erkennen. Die Lösungsgüte der TS kann nicht mit der der VNS konkurrieren. Es wird zwar eine vergleichsweise hohe Lösungsqualität erzielt, die jedoch nicht an die zuvor betrachteten Verfahren heranreicht. Es scheint, dass die hier verwendete TS mit weniger stark strukturierten Probleminstanzen weniger gut zurechtkommt. Allerdings sollte noch angemerkt werden, dass die TS_{CPU} bei den Gehring-Instanzen ebenfalls neue beste Lösungen findet, die von der Problemklasse her betrachtet weitestgehend denen der $VNS \times SimA_{CPU}$ entsprechen. Mit Instanz R1_600 findet die Methode im Vergleich zur $VNS \times SimA_{CPU}$ noch eine weitere neue beste Lösung.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	12,95	0,00	0,00
c2	835,26	844,04	837,41	28,56	1,05	0,26
c3	826,14	828,07	827,39	47,20	0,23	0,15
c4	1028,42	1029,42	1028,78	582,46	0,10	0,03
c5	1291,29	1309,77	1302,54	157,80	1,43	0,87
c11	1042,11	1042,11	1042,11	232,62	0,00	0,00
c12	819,56	819,56	819,56	1,61	0,00	0,00
Summe	6367,39	6397,58	6382,40	1063,20	0,47	0,24

Tabelle 4.11: Lösungen der TS_{CPU} auf den Christofides-Instanzen

Zusammenfassend lässt sich aufgrund der Ergebnisse in den nicht strukturierten bzw. weniger strukturierten Problemen die Empfehlung für die $VNS \times SimA_{CPU}$ aussprechen. Eine etwaige Begründung könnte in dem Annehmen der Moves liegen. In der $VNS \times SimA_{CPU}$ entscheidet die Annahmewahr-

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1630,59	1619,69	15,24	0,76	0,08
T75b	1344,62	1346,14	1345,07	5,41	0,11	0,03
T75c	1291,01	1291,03	1291,01	9,39	0,00	0,00
T75d	1365,42	1375,06	1368,01	29,54	0,71	0,19
T100a	2041,34	2057,90	2048,40	109,75	0,81	0,35
T100b	1939,90	1941,45	1940,70	95,68	0,08	0,04
T100c	1406,20	1421,66	1415,28	107,95	1,10	0,65
T100d	1580,46	1605,96	1598,41	53,11	1,61	1,14
T150a	3055,23	3158,43	3108,29	49,38	3,38	1,74
T150b	2727,20	2775,74	2737,34	349,36	1,78	0,37
T150c	2341,84	2366,43	2364,22	98,26	1,05	0,96
T150d	2645,40	2665,08	2661,12	21,59	0,74	0,59
T385	24366,69	25027,91	24677,67	1004,85	2,71	1,28
Summe	47723,67	48663,38	48175,20	1949,52	1,97	0,95

Tabelle 4.12: Lösungen der TS_{CPU} auf den Taillard-Instanzen

scheinlichkeit, ob ein Move angenommen wird, d.h., gerade bei geringen Temperaturen werden Moves, die die Lösung sehr stark verschlechtern, nicht akzeptiert. Das ist im Falle der TS nicht gegeben. Hier muss der Move, der nicht tabu ist, ausgeführt werden - unabhängig davon wie stark er die aktuelle Lösung verschlechtert. Das könnte dazu führen, dass sich die TS weniger lange in einem „guten“ Bereich des Lösungsraumes aufhält und dadurch dort nicht alle Punkte erreicht, da sie zuvor schon wieder an anderer Stelle weitersuchen muss.

4.2.2 Analyse GPU-basierter Metaheuristiken mit geringem CPU-Anteil

In diesem Abschnitt sollen die Metaheuristiken bzw. die Varianten der Metaheuristiken, die intensiv von der GPU Gebrauch machen, betrachtet werden. Bevor die eigentlichen Ergebnisse der Probleminstanzen präsentiert werden können, muss zunächst die Anzahl der Threads pro Block ermittelt werden, da die Resultate, die für die Operatoren allein betrachtet, gezeigt werden, nicht mehr der angepassten Implementierung entsprechen müssen.

4.2.2.1 Bestimmung der Threadanzahl pro Block

Zur Bestimmung der optimalen Threadanzahl werden die entsprechenden Kernel jeweils 10 Mal pro Probleminstanz (Größe zwischen 100 und 1002 Knoten) und Threadanzahl (32, 64, 128, 256, 512) aufgerufen.[106] Des Weiteren ist die Anzahl der Lösungen, die parallel bearbeitet werden, auf 16 Lösungen konstant gehalten.

Letzteres lässt sich direkt aus Abbildung 4.19 begründen. Hierbei wird der Algorithmus $VNS \times SimA_{GPU}$ 100 Mal gestartet. Dabei wird zu Beginn die Anzahl der Lösungen, die parallel bearbeitet werden, zufällig zwischen 1 und 33 gewählt. Danach wird der Algorithmus jeweils mit der Threadanzahl 32, 64, 128, 256 und 512 gestartet und die Kernelzeit für jeweils eine Iteration auf der GPU gemessen. Es lässt sich erkennen[107], dass die Kernelzeiten für eine Anzahl von Lösungen bis 16 relativ ähnlich sind. Ab der 17. Lösung ist ein Sprung der Kernelzeit zu erkennen, die dann wiederum bis zu 32 Lösungen konstant bleibt. Das Verhalten lässt sich direkt durch die Hardwarearchitektur der GPU erklären. Jeder

[106] Hier ist es wieder notwendig, dass die Threadanzahl einer 2er-Potenz entspricht, da die Bestimmung des besten Moves direkt in den Kernel, der sämtliche Arbeit übernimmt, integriert ist und dies Voraussetzung dafür ist, dass der korrekte beste Move bestimmt werden kann. Weiterhin ist es nicht möglich 1024 Threads pro Kernel zu starten, da dies die Ressourcen der Grafikkarte übersteigt.

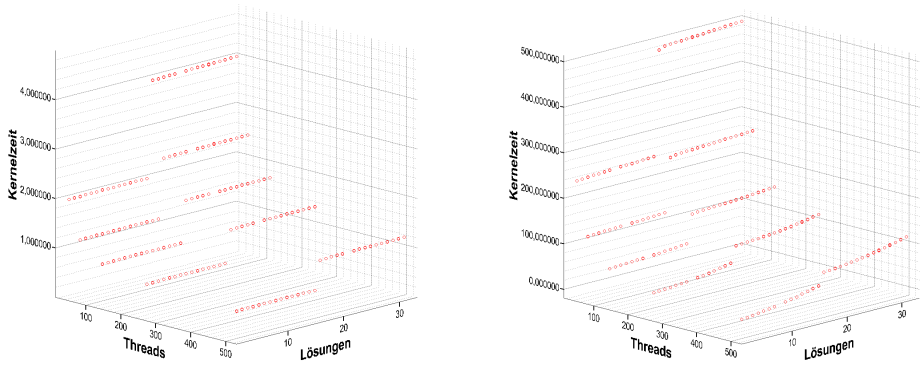
[107] Auch bei anderen Knotenzahlen ergeben sich analoge Verläufe der Kernelzeiten.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2568,90	**2567,67**	579,35	-0,06	**-0,11**
C1_400	6765,03	6740,02	**6734,11**	2236,63	-0,37	**-0,46**
C1_600	13465,24	13478,15	**13452,24**	2784,96	0,10	**-0,10**
C1_800	23999,35	23845,94	**23744,66**	3085,27	-0,64	**-1,06**
C1_1000	39811,15	39712,12	**39609,58**	3524,05	-0,25	**-0,51**
C2_200	1368,89	1492,96	1488,38	3320,18	9,06	8,73
C2_400	2799,40	3182,86	3173,43	2068,76	13,70	13,36
C2_600	5441,98	6318,55	6301,32	2759,48	16,11	15,79
C2_800	8241,83	9748,10	9668,65	3593,85	18,28	17,31
C2_1000	11792,66	14558,69	14405,01	3568,65	23,46	22,15
R1_200	2878,24	2881,02	2879,01	1469,54	0,10	0,03
R1_400	7192,03	7206,80	7193,39	954,96	0,21	0,02
R1_600	15691,37	15717,47	**15625,16**	3588,87	0,17	**-0,42**
R1_800	27650,67	27917,21	27784,29	3575,59	0,96	0,48
R1_1000	42311,91	43298,48	43101,07	3148,51	2,33	1,87
R2_200	1610,30	1631,77	1625,51	2099,09	1,33	0,94
R2_400	3323,05	3383,98	3352,50	3264,62	1,83	0,89
R2_600	6254,17	6360,48	6299,72	2315,83	1,70	0,73
R2_800	10230,06	10626,57	10389,16	3292,45	3,88	1,56
R2_1000	15149,40	15872,26	15658,62	3531,52	4,77	3,36
RC1_200	2804,20	2844,06	2825,54	708,72	1,42	0,76
RC1_400	7381,47	7420,90	**7360,85**	2680,62	0,53	**-0,28**
RC1_600	14786,15	15043,84	15006,56	1526,96	1,74	1,49
RC1_800	26720,23	27423,76	27025,62	3313,23	2,63	1,14
RC1_1000	41583,54	42355,96	42128,18	3597,17	1,86	1,31
RC2_200	1513,00	1527,84	1513,00	414,71	0,98	0,00
RC2_400	3100,06	3149,93	3116,26	2389,47	1,61	0,52
RC2_600	5732,08	5963,77	5882,77	2268,07	4,04	2,63
RC2_800	9217,99	9849,11	9714,06	1886,17	6,85	5,38
RC2_1000	13631,76	14989,57	14213,28	3466,56	9,96	4,27
Summe	375017,70	387111,03	383839,60	77013,84	1,86	0,99

Tabelle 4.13: Lösungen der TS_{CPU} auf den Gehring-Instanzen

Block bearbeitet eine Lösung. Durch die Restriktion, dass jeder Block genau einem SM zugeordnet wird, resultiert, dass bei einem Start des Verfahrens mit weniger als 16 Lösungen, Blöcke - sprich SMs - brach liegen. Wenn demnach 16 Lösungen verwendet werden, werden diese parallel abgearbeitet. Ähnlich verhält es sich, wenn die Anzahl der Lösungen 16 übersteigt. Angenommen, es werden 17 Lösungen gestartet, dann ist in diesem Fall jeder Multiprozessor mit einem Block und einer mit zwei Blöcken beschäftigt. Während dieser seinen zweiten Block bearbeitet, müssen die anderen warten bzw. liegen wieder brach. Dieses Verhalten ließe sich „beliebig“ fortführen. Daraus resultiert bei der Wahl der Anzahl der Lösungen, die parallel bearbeitet werden, dass sie einem Vielfachen der Anzahl der SMs, über die die GPU verfügt, entsprechen sollten, d.h. im Falle der Nvidia Geforce 580 GTX 16, 32, 48, 64 usw.. Deshalb wird in dieser Arbeit mit 16 Lösungen parallel gearbeitet. Diese Anzahl bildet ebenfalls die Grundlage zur Bestimmung der Anzahl der Threads, die pro Kernel gestartet und im Folgenden betrachtet werden.

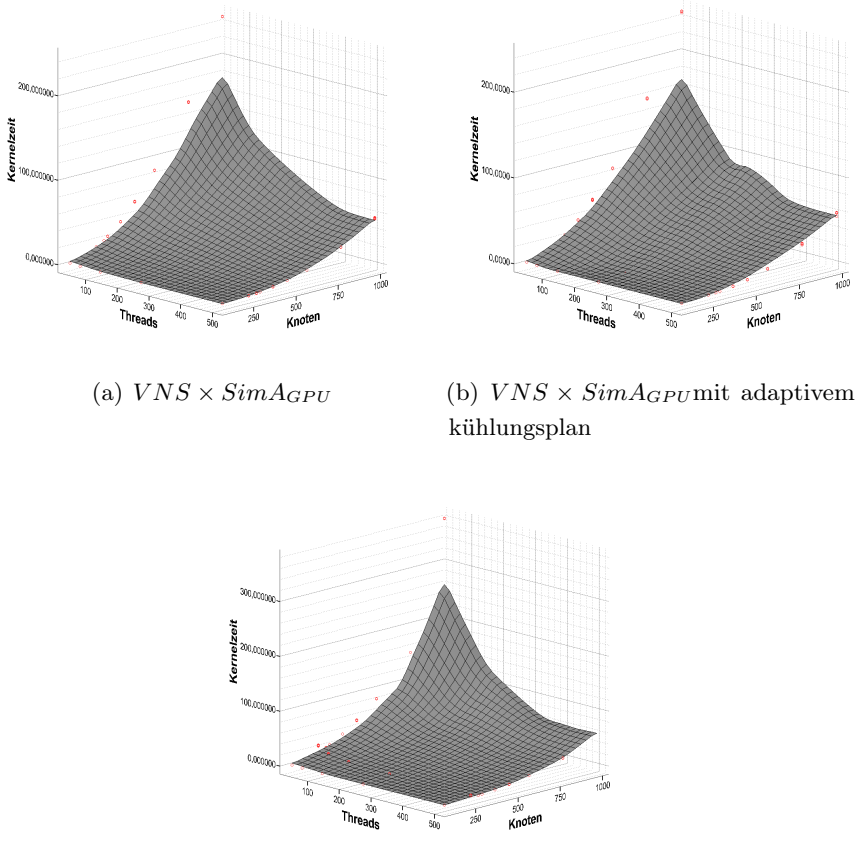
Nachdem nun die Anzahl der Lösungen, die parallel bearbeitet werden, festgelegt ist, finden sich in Abbildung 4.20 die unterschiedlichen Auswirkungen der Thread- und Knotenzahl im Kontext der $VNS \times SimA_{GPU}$. Betrachtet werden dabei die $VNS \times SimA_{GPU}$, $VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan sowie die $VNS \times SimA_{GPU}$ mit COSA. Hintergrund dieser Wahl ist die Tatsache, dass sich sowohl der adaptive Abkühlungsplan, der auf der GPU verwaltet wird, auf die Kernelzeit auswirkt als auch COSA im Shaking-Verfahren, das ebenfalls auf der GPU durchgeführt wird. Bei allen drei Verfahren zeigt sich der gleiche Verlauf der Kernelzeiten. Mit steigender Knotenzahl steigt die Ausführungsdauer. Bei steigender Threadanzahl sinkt die Kernelzeit erheblich, d.h., in diesem Fall ist eindeutig festzustellen, dass mit der größtmöglichen Threadzahl pro Block, also 512, in allen Fällen gearbeitet werden sollte. Somit kann man hier auf eine individuelle Anpassung der Threadanzahl, so



(a) 101 Knoten (b) 1001 Knoten

Abbildung 4.19: Kernelzeit in Abhängigkeit der Thread- und Lösungsanzahl

wie sie in der Implementierung der Operatoren umgesetzt wird, verzichten. Weiterhin zeigt sich, dass der adaptive Abkühlungsplan und COSA Auswirkungen auf die Dauer des Kernels haben. Diese sind jedoch sehr gering, sodass sie keiner weiteren Analyse bedürfen.

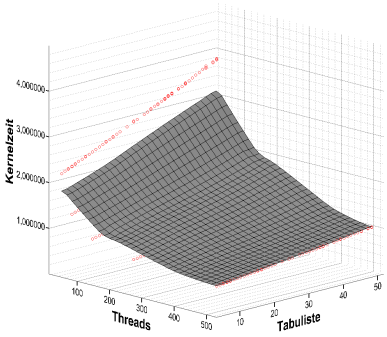


(a) $VNS \times SimA_{GPU}$ (b) $VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan

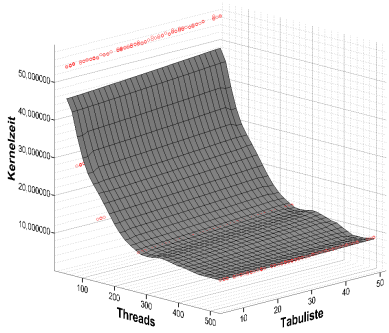
(c) $VNS \times SimA_{GPU}$ mit COSA

Abbildung 4.20: Kernelzeit in Abhängigkeit der Thread- und Knotenanzahl

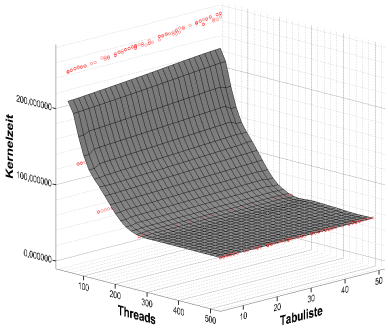
Nun gilt es noch die TS_{GPU} zu beleuchten. Hierbei werden drei Instanzen, die 101, 484 und 1001 Knoten enthalten, als Repräsentanten selektiert. Weiterhin werden auch hier 100 Durchläufe gestartet. Dabei wird pro Durchlauf zufällig die Größe der Tabuliste zwischen 4 und 51 gezogen. Mit dieser Größe wird dann der Algorithmus in der Konfiguration mit 32, 64, 128, 256 und 512 Threads mit jeweils zwischen 100 und 1002 Knoten gestartet. Abbildung 4.21 zeigt den Verlauf der Kernelzeiten. Es ist eindeutig ersichtlich, dass mit steigender Threadzahl die Kernelzeit rapide abnimmt. Das gilt über alle Problemgrößen hinweg. Weiterhin kann man erkennen, dass die Größe der Tabuliste mit zunehmender Knotenanzahl weniger Einfluss auf die gesamte Kernelzeit nimmt. Das lässt sich damit begründen, dass bei kleiner Knotenanzahl ein größerer relativer Zusammenhang mit der Länge der Tabuliste besteht, wohingegen bei großer Knotenanzahl der Zeitanteil zur Prüfung des Tabustatus geringer ist.



(a) 101 Knoten



(b) 484 Knoten



(c) 1001 Knoten

Abbildung 4.21: Kernelzeit in Abhängigkeit der Threadanzahl und Tabulistenlänge (TS_{GPU})

Zusammenfassend lässt sich sagen, dass im Folgenden die größtmögliche Anzahl an Threads pro Block selektiert wird. Weiterhin wird die Anzahl der parallel zu bearbeitenden Lösungen auf 16 gesetzt.

4.2.2.2 Betrachtung der $VNS \times SimA_{GPU}$

Multistart-$VNS \times SimA_{GPU}$

Zunächst wird die $VNS \times SimA_{GPU}$-Version betrachtet, bei der das gesamte Verfahren der $VNS \times SimA_{CPU}$ parallelisiert wird und als Daten die besten Lösungen zwischen den Blöcken ausgetauscht

werden. Hierbei wird für jeden Block bzw. Lösung, eine Initialtemperatur von 100 gesetzt, die nach jeder Anwendung eines Moves mit dem Faktor 0,9999 abgekühlt wird. Jeder Block bekommt dabei eine unterschiedliche Startlösung, die wie beschrieben mittels der Heuristik von Clarke und Wright (1964) und dem Framework von Groër et al. (2010) generiert werden, zugewiesen. Auch im Falle des Multistart wird eine Population von Lösungen verwaltet, die nach jedem abgeschlossenen Kernelaufruf auf der CPU aktualisiert wird. Dabei werden auf der GPU jeweils 100 Iterationen ausgeführt, wobei eine Iteration die Betrachtung aller Moves und deren sequentielle Anwendung auf die aktuelle Lösung beinhaltet. Nach dieser Iteration auf der GPU wird von allen erzeugten Lösungen die beste ausgewählt und in die vorhandene Population eingefügt. Dabei wird eine zufällige alte Lösung entfernt; allerdings wird immer garantiert, dass die beste Lösung, die bisher gefunden wurde, nicht ersetzt wird. Danach arbeiten die Blöcke der GPU weiterhin an ihrer Lösung. Nach 100 GPU-Iterationen, d.h. insgesamt 10000 Iterationen, die auf der GPU ausgeführt wurden, wird an alle Blöcke die bisher beste Lösung gesendet und alle arbeiten an dieser Stelle mit ihrem jeweiligen Temperaturniveau weiter. Das Temperaturniveau wird dabei über die Kernelaufrufe hinweg gespeichert, sodass der folgende Aufruf mit der Temperatur weiterarbeiten kann, mit der der vorangegangene geschlossen hat. Auch hier findet ein Aufwärmen der Temperatur auf 5 statt, wenn 0,2 unterschritten ist. Das Verfahren endet ebenfalls in Abhängigkeit der Knotenanzahl nach einer zuvor festgelegten Zeit.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	583,90	582,82	2664,51	0,72	0,54
g10	736,26	741,56	741,02	3459,34	0,72	0,65
g11	912,84	919,12	918,19	3390,05	0,69	0,59
g12	1102,69	1111,31	1110,22	2431,20	0,78	0,68
g13	857,19	860,91	859,53	1592,19	0,43	0,27
g14	1080,55	1081,28	1080,90	2566,66	0,07	0,03
g15	1337,92	1346,71	1345,42	2706,89	0,66	0,56
g16	1612,50	1628,65	1622,67	2999,34	1,00	0,63
g17	707,76	708,14	707,90	3585,52	0,05	0,02
g18	995,13	1002,47	1001,71	3022,64	0,74	0,66
g19	1365,60	1370,85	1369,17	3296,43	0,38	0,26
g20	1818,25	1830,98	1828,91	3420,14	0,70	0,59
Summe	13106,40	13185,89	13168,44	35134,91	0,61	0,47

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	586,41	1,16	584,09	0,75	583,67	0,68
g10	743,45	0,98	741,93	0,77	741,21	0,67
g11	923,16	1,13	918,98	0,67	918,72	0,64
g12	1121,21	1,68	1115,23	1,14	1110,42	0,70
g13	864,45	0,85	859,53	0,27	859,53	0,27
g14	1087,57	0,65	1081,81	0,12	1081,24	0,06
g15	1353,94	1,20	1348,82	0,81	1346,44	0,64
g16	1641,98	1,83	1629,77	1,07	1624,36	0,74
g17	708,93	0,16	707,97	0,03	707,91	0,02
g18	1007,11	1,20	1004,19	0,91	1002,71	0,76
g19	1379,38	1,01	1373,78	0,60	1370,78	0,38
g20	1846,59	1,56	1834,53	0,90	1831,49	0,73
Summe	13264,19	1,20	13200,62	0,72	13178,46	0,55

Tabelle 4.14: Lösungen der Multistart-$VNS \times SimA_{GPU}$ auf den Golden-Instanzen

Wie in Tabelle 4.14 zu erkennen ist, wird die Lösungsgüte durch das Auslagern des Verfahrens auf die GPU verbessert. So beträgt die Gesamtabweichung von den besten bekannten Lösungen nur noch 0,47%. Weiterhin kann die durchschnittliche Abweichung um über 0,1%-Punkte reduziert werden, wodurch man schlussfolgern kann, dass der Algorithmus auf der GPU robustere Ergebnisse liefert. Ferner erreicht das Multistart-Verfahren bereits nach kurzer Zeit bessere Ergebnisse auf den Golden-Instanzen (Abweichung von 1,2% nach 2 Minuten). Erklärbar sind die besseren Ergebnisse vornehmlich durch die Anzahl der Lösungen, die parallel bearbeitet werden. Hier werden pro Durchlauf prinzipiell 16 Lösungen generiert. Um dies mit der $VNS \times SimA_{CPU}$ zu erreichen, müsste man sie 16 Mal starten.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	18,68	0,00	0,00
c2	835,26	836,55	835,26	6,51	0,15	0,00
c3	826,14	826,14	826,14	77,21	0,00	0,00
c4	1028,42	1029,82	1028,78	504,50	0,14	0,03
c5	1291,29	1300,68	1293,13	519,39	0,73	0,14
c11	1042,11	1042,15	1042,11	494,58	0,00	0,00
c12	819,56	819,56	819,56	73,18	0,00	0,00
Summe	6367,39	6379,51	6369,59	1694,05	0,19	0,03

Tabelle 4.15: Lösungen der Multistart-$VNS \times SimA_{GPU}$ auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1620,89	1620,81	11,64	0,16	0,15
T75b	1344,62	1344,85	1344,62	23,79	0,02	0,00
T75c	1291,01	1291,08	1291,01	6,47	0,01	0,00
T75d	1365,42	1365,42	1365,42	6,01	0,00	0,00
T100a	2041,34	2049,02	2045,80	71,49	0,38	0,22
T100b	1939,90	1941,06	1940,72	105,51	0,06	0,04
T100c	1406,20	1406,95	1406,20	70,45	0,05	0,00
T100d	1580,46	1583,17	1581,26	106,23	0,17	0,05
T150a	3055,23	3056,80	3055,23	572,46	0,05	0,00
T150b	2727,20	2733,87	2728,17	455,83	0,24	0,04
T150c	2341,84	2361,59	2361,44	570,24	0,84	0,84
T150d	2645,40	2660,55	2659,16	465,58	0,57	0,52
T385	24366,69	24627,22	24555,36	3477,00	1,07	0,77
Summe	47723,67	48042,47	47955,19	5942,71	0,67	0,49

Tabelle 4.16: Lösungen der Multistart-$VNS \times SimA_{GPU}$ auf den Taillard-Instanzen

Die Tabellen 4.15, 4.16 und 4.17 belegen die Überlegenheit der $VNS \times SimA_{GPU}$ im Vergleich zu den Metaheuristiken, die einen hohen Berechnungsanteil auf der CPU ausführen. Bei den Christofides-Instanzen erreicht das Verfahren eine Abweichung von nur 0,03%. Ebenso ist die Abweichung bei den Taillard-Instanzen geringer als die zuvor betrachteten Verfahren. Das gleiche gilt auch für die Instanzen von Gehring und Homberger (1999) bei denen eine mittlere Abweichung der besten gefundenen Lösungen zu den besten bekannten Lösungen von 0,46% vorliegt. Der Algorithmus hat zehn bisher beste bekannte Lösungen verbessern können, was seine Stärke nochmals beweist. Alles in allem lässt sich bis hier schlussfolgern, dass die extensive Nutzung der GPU eine deutliche Steigerung der Lösungsgüte zur Folge hat.

Multistart-$VNS \times SimA_{GPU}$ mit adaptiver Abkühlung

Nachdem im vorangegangenem Abschnitt die Stärke der Algorithmen bei Nutzung der GPU gezeigt wurde, soll auf die Auswirkungen einer adaptiven Abkühlung - insbesondere im Hinblick auf die Lösungsgüte - eingegangen werden. Grundsätzlich läuft das Verfahren mit den gleichen Parametern

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,77	**2567,67**	131,54	-0,11	**-0,11**
C1_400	6765,03	6735,34	**6728,92**	3434,87	-0,44	**-0,53**
C1_600	13465,24	13452,32	**13449,25**	3024,42	-0,10	**-0,12**
C1_800	23999,35	23743,84	**23726,37**	3316,75	-1,06	**-1,14**
C1_1000	39811,15	39528,68	**39455,07**	3479,51	-0,71	**-0,89**
C2_200	1368,89	1485,45	1483,59	2371,27	8,51	8,38
C2_400	2799,40	3143,79	3132,58	2777,89	12,30	11,90
C2_600	5441,98	6325,37	6318,06	3482,22	16,23	16,10
C2_800	8241,83	9683,31	9658,25	2500,47	17,49	17,19
C2_1000	11792,66	14364,40	14320,04	3639,03	21,81	21,43
R1_200	2878,24	2875,35	**2872,02**	2239,42	-0,10	**-0,22**
R1_400	7192,03	7209,59	**7186,31**	3598,34	0,24	**-0,08**
R1_600	15691,37	15643,72	**15618,17**	3281,95	-0,30	**-0,47**
R1_800	27650,67	27918,51	27897,53	3536,73	0,97	0,89
R1_1000	42311,91	42959,08	42936,90	1789,78	1,53	1,48
R2_200	1610,30	1625,23	1610,30	2459,37	0,93	0,00
R2_400	3323,05	3380,35	3373,99	3393,26	1,72	1,53
R2_600	6254,17	6315,27	6288,14	3504,79	0,98	0,54
R2_800	10230,06	10482,31	10463,03	840,19	2,47	2,28
R2_1000	15149,40	15428,27	15317,36	3585,19	1,84	1,11
RC1_200	2804,20	2815,38	**2803,34**	3285,60	0,40	**-0,03**
RC1_400	7381,47	7333,68	**7311,44**	3594,04	-0,65	**-0,95**
RC1_600	14786,15	14911,94	14854,92	3367,92	0,85	0,47
RC1_800	26720,23	26914,17	26836,10	3575,17	0,73	0,43
RC1_1000	41583,54	41990,07	41942,25	3347,01	0,98	0,86
RC2_200	1513,00	1519,16	1514,74	683,10	0,41	0,11
RC2_400	3100,06	3137,41	3123,10	3255,88	1,20	0,74
RC2_600	5732,08	5864,29	5820,80	2939,54	2,31	1,55
RC2_800	9217,99	9539,35	9417,12	1428,83	3,49	2,16
RC2_1000	13631,76	14022,67	13839,47	3589,90	2,87	1,52
Summe	375017,70	382916,07	381866,83	85453,98	0,74	0,46

Tabelle 4.17: Lösungen der Multistart-$VNS \times SimA_{GPU}$ auf den Gehring-Instanzen

ab, wie das zuvor beschriebene. Unterschiede ergeben sich lediglich in der Wahl der Starttemperatur sowie in der Bestimmung des Zeitpunktes, wann abgekühlt wird. Dabei wird die Starttemperatur in Abhängigkeit vom Zielfunktionswert der Initiallösung, die dem Block zugeteilt wird, gewählt. Sie wird so gewählt, dass eine Verschlechterung des Zielfunktionswertes zwischen 2% und 5% (zufällig gewählt) mit einer Wahrscheinlichkeit zwischen 50% und 80% (zufällig gewählt) angenommen wird. Damit ist sichergestellt, dass jeder Block mit einer individuellen Starttemperatur beginnt. Diese Konfiguration hat sich in vorangegangenen Tests als sehr gut erwiesen. Weiterhin entscheidet der Verlauf des Suchprozesses an sich, ob die Temperatur angepasst wird oder nicht. Dazu werden die letzten 20 Moves betrachtet. Wenn der gleitende Durchschnitt über die Zielfunktionsveränderungen c_Δ der letzten 20 Moves größer gleich 0 ist, findet eine Temperaturanpassung statt. Mit dem so erreichten Temperaturniveau wird für mindestens 20 weitere Moves gearbeitet. Alle anderen Parameter entsprechen vollständig denen des zuvor beschriebenen Verfahrens. So findet auch hier bspw. ein Aufwärmen der Temperatur, wenn sie ein bestimmtes Niveau unterschritten hat, statt.

In Tabelle 4.18 sind die Ergebnisse des Verfahrens dargestellt. Es zeigt sich, dass die Endergebnisse nach 3600 Sekunden weitestgehend denen des Multistart-$VNS \times SimA_{GPU}$ entsprechen. Auch hier liegt eine äußerst hohe Lösungsgüte vor. Unterschiede finden sich jedoch bei Betrachtung des Zeitverlaufs. So beläuft sich die mittlere Abweichung nach 2 Minuten auf über 2,5%, wohingegen die des reinen Multistart-Verfahrens bei 1,2% liegt. Das könnte ein Zeichen dafür sein, dass sich das Verfahren zu lange auf einem Temperaturniveau aufhält und dadurch unnötige Iterationen durchführt und es somit länger dauert, bis es zum lokalen Optimum konvergiert. Bei weiterem Einsatz des Verfahrens müsste dementsprechend eine Parameteroptimierung durchgeführt werden, hinsichtlich der Frage, wann und um welchen Faktor abgekühlt werden sollte.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	584,09	582,41	3183,26	0,76	0,47
g10	736,26	740,87	739,96	2710,46	0,63	0,50
g11	912,84	918,93	917,32	3051,86	0,67	0,49
g12	1102,69	1113,21	1112,23	3351,69	0,95	0,86
g13	857,19	860,60	859,19	1852,44	0,40	0,23
g14	1080,55	1081,27	1080,90	3552,30	0,07	0,03
g15	1337,92	1347,52	1345,78	2799,55	0,72	0,59
g16	1612,50	1628,30	1624,07	3568,10	0,98	0,72
g17	707,76	708,56	707,93	3394,01	0,11	0,02
g18	995,13	1003,05	1001,29	2921,57	0,80	0,62
g19	1365,60	1371,70	1370,77	3441,85	0,45	0,38
g20	1818,25	1831,58	1829,13	3537,91	0,73	0,60
Summe	13106,40	13189,70	13170,96	37365,01	0,64	0,49

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	598,34	3,21	584,64	0,85	582,52	0,49
g10	755,21	2,57	745,41	1,24	740,69	0,60
g11	934,99	2,43	929,56	1,83	920,20	0,81
g12	1139,59	3,35	1139,59	3,35	1117,60	1,35
g13	885,62	3,32	862,78	0,65	859,19	0,23
g14	1117,56	3,42	1087,52	0,65	1081,17	0,06
g15	1384,24	3,46	1372,92	2,62	1348,84	0,82
g16	1667,39	3,40	1667,39	3,40	1634,46	1,36
g17	717,06	1,31	709,98	0,31	708,12	0,05
g18	1018,11	2,31	1013,38	1,83	1002,10	0,70
g19	1383,58	1,32	1381,56	1,17	1372,26	0,49
g20	1864,81	2,56	1850,80	1,79	1836,66	1,01
Summe	13466,48	2,75	13345,52	1,82	13203,82	0,74

Tabelle 4.18: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Golden-Instanzen

Im Anhang findet man in den Tabellen B.16, B.17 und B.18 die Resultate für die Christofides-, Taillard- und Gehring-Instanzen. Hier zeigt sich, dass der Algorithmus besonders bei kleinen Instanzen, bei denen mit einer recht geringen Ausführungszeit gearbeitet wird, schlechtere Ergebnisse erzielt als der Pendant-Algorithmus ohne den adaptiven Abkühlungsplan. Auch dies lässt die Vermutung zu, dass gerade für kurze Zeiten der Abkühlungsplan optimiert werden sollte. Auch hier werden im Rahmen der Gehring-Instanzen neue beste Lösungen gefunden (8 an der Zahl). Allerdings beträgt die mittlere Abweichung der besten gefundenen Lösung zur besten bekannten Lösung etwa 1%.

Der Algorithmus mit adaptivem Abkühlungsplan liegt also bei einer genügend langen Laufzeit auf dem Niveau des Verfahrens ohne adaptivem Abkühlungsplan. Bei kürzeren Laufzeiten ist zunächst eine weitere Parameteroptimierung notwendig, um auf ein besseres Ergebnisniveau zu gelangen.

Multistart-$VNS \times SimA_{GPU}$ mit COSA

Die einzige Veränderung, die bei $VNS \times SimA_{GPU}$ mit COSA, im Vergleich zur Basisversion vorgenommen wird, ist der Austausch des Shaking-Verfahrens. Es wird während des Shakings zufällig ein Blockpartner bzw. eine Partnerlösung selektiert, die als Informationsspender dient. Auf dieser Basis wird dann der mögliche Cross-Exchange-Move ermittelt unter der Bedingung, dass die Nachbarschaftsbeziehung des Informationsspenders erfüllt wird. Ist ein solcher Move möglich, wird er ausgeführt. Wenn kein gültiger Move gefunden wird, wird dementsprechend keine Shaking-Operation ausgeführt. Weiterhin wird nach jeder Iteration in der alle Operatoren begutachtet werden, die aktuelle Lösung von jedem Block in den Globalspeicher geschrieben, damit die anderen Blöcke auf den Informationsspender zugreifen können.

Die Ergebnisse der Golden-Instanzen zeigen, dass es im Vergleich zu den bisher betrachteten reinen

GPU-Verfahren zu einer minimalen Verschlechterung der Lösungsgüte kommt (vgl. Tabelle 4.19). Die Verschlechterung lässt sich eventuell damit erklären, dass durch die Restriktion des COSA, durch die nur bestimmte Nachbarn beim Shaking betrachtet werden dürfen, seltener ein Shaking-Move ausgeführt wird, da seltener ein gültiger gefunden wird.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	583,08	580,78	3292,67	0,58	0,19
g10	736,26	740,99	740,83	2783,02	0,64	0,62
g11	912,84	918,74	918,04	2333,44	0,65	0,57
g12	1102,69	1111,22	1109,03	3124,63	0,77	0,57
g13	857,19	861,14	859,61	1676,78	0,46	0,28
g14	1080,55	1082,40	1081,96	1633,68	0,17	0,13
g15	1337,92	1346,18	1345,33	3451,90	0,62	0,55
g16	1612,50	1627,15	1625,41	3592,04	0,91	0,80
g17	707,76	708,91	708,60	3473,83	0,16	0,12
g18	995,13	1004,16	1003,14	2200,21	0,91	0,80
g19	1365,60	1373,81	1371,41	2851,65	0,60	0,43
g20	1818,25	1837,79	1835,77	3298,59	1,07	0,96
Summe	13106,40	13195,57	13179,91	33712,42	0,68	0,56

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	586,74	1,21	584,95	0,90	583,00	0,57
g10	743,74	1,02	741,11	0,66	740,83	0,62
g11	925,00	1,33	919,73	0,75	918,39	0,61
g12	1120,86	1,65	1116,90	1,29	1110,00	0,66
g13	865,25	0,94	862,68	0,64	859,61	0,28
g14	1086,17	0,52	1083,37	0,26	1081,96	0,13
g15	1356,03	1,35	1349,37	0,86	1345,63	0,58
g16	1642,56	1,86	1635,43	1,42	1626,18	0,85
g17	710,54	0,39	709,13	0,19	709,13	0,19
g18	1009,73	1,47	1005,96	1,09	1003,71	0,86
g19	1379,45	1,01	1376,04	0,76	1371,64	0,44
g20	1851,99	1,86	1843,14	1,37	1838,55	1,12
Summe	13278,05	1,31	13227,80	0,93	13188,64	0,63

Tabelle 4.19: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Golden-Instanzen

Bei Betrachtung der anderen Problemklassen erweist sich die $VNS \times SimA_{GPU}$ mit COSA insbesondere im Hinblick auf die Gehring-Instanzen als sehr erfolgreich (vgl. Tabellen B.19, B.20 und B.21). Hier findet der Algorithmus insgesamt 11 neue beste Lösungen und liegt mit einer mittleren Abweichung der besten gefundenen Lösungen zu den besten bekannten Lösungen von 0,37% deutlich unter den bisher vorgestellten Verfahren. Eine mögliche Erklärung ist, dass durch stark strukturierte Probleme die Diversifizierung durch COSA sehr große Sprünge im Lösungsraum vornimmt, die durch die geclusterte Kundenverteilung entstehen. Diese starken Sprünge zerstören eventuell zu große Teile der Lösung bzw. lenken sie in weniger gute Bereiche des Lösungsraumes. Demgegenüber liegt bei den weniger stark strukturierten Problemen wie bei den R-Instanzen von Gehring und Homberger (1999) eine uniforme Verteilung der Knoten vor. Hier wirken sich Sprünge durch den Lösungsraum eventuell weniger stark aus. Auch bei den Christofides- und Taillard-Instanzen erzielt der Algorithmus mit COSA als Shaking-Verfahren sehr gute Resultate.

4.2.2.3 Betrachtung der TS_{GPU}

Bisher liegt der Fokus vornehmlich auf dem Verfahren $VNS \times SimA_{GPU}$ und ihren Derivaten. Nun sollen die Ergebnisse, die die TS auf der GPU erzielt, näher analysiert werden. Der Algorithmus entspricht zu großen Teilen der TS_{CPU}. Allerdings wird hier pro Lösung, die parallel bearbeitet wird, eine individuelle Tabuliste mit individuellen unteren und oberen Schranken verwaltet. Durch das Shaking-Verfahren ist bereits keine Deterministik mehr in dem Algorithmus gegeben (das Shaking wird

analog zur TS_{CPU} ebenfalls nur nach einer bestimmten Anzahl an Iterationen ausgeführt). Diese wird durch die unterschiedlichen Tabulisten noch weiter aufgelöst. Die Länge der Tabuliste wird zu Beginn des Verfahrens für jeden Block zufällig zwischen 4 und 16 gezogen. Die untere und obere Schranke wird hingegen zunächst für alle Blöcke mit dem gleichen Wert von jeweils 0,0001 initialisiert und entsprechend den Beschreibungen der TS_{CPU} während des Suchverlaufs angepasst. Auch hier findet ein Lösungsaustausch statt, der in Analogie zur CPU-Variante abläuft. Das bedeutet, wenn nach 10000 Iterationen ein Block keine neue beste Lösung im Vergleich mit allen anderen Blöcken gefunden hat, so wird die aktuell beste Lösung, die von allen gefunden wurde, als Lösung zur weiteren Bearbeitung gesetzt.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	584,23	583,48	2519,79	0,78	0,65
g10	736,26	742,14	741,06	3137,33	0,80	0,65
g11	912,84	918,69	915,62	3419,64	0,64	0,30
g12	1102,69	1114,66	1112,94	2343,76	1,09	0,93
g13	857,19	862,88	859,65	3302,81	0,66	0,29
g14	1080,55	1081,26	1081,02	3577,51	0,07	0,04
g15	1337,92	1348,19	1344,70	2855,79	0,77	0,51
g16	1612,50	1627,52	1623,17	3369,39	0,93	0,66
g17	707,76	708,83	708,22	1508,23	0,15	0,07
g18	995,13	1002,46	1000,03	1697,78	0,74	0,49
g19	1365,60	1371,51	1369,56	2492,73	0,43	0,29
g20	1818,25	1838,04	1829,94	2998,73	1,09	0,64
Summe	13106,40	13200,43	13169,41	33223,49	0,72	0,48

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	584,69	0,86	584,57	0,84	583,56	0,66
g10	743,18	0,94	741,88	0,76	741,86	0,76
g11	923,93	1,21	919,44	0,72	917,24	0,48
g12	1121,21	1,68	1116,10	1,22	1114,46	1,07
g13	863,69	0,76	860,97	0,44	859,81	0,31
g14	1084,47	0,36	1081,75	0,11	1081,17	0,06
g15	1356,61	1,40	1349,20	0,84	1346,32	0,63
g16	1642,81	1,88	1633,15	1,28	1628,99	1,02
g17	709,90	0,30	708,94	0,17	708,22	0,07
g18	1007,09	1,20	1002,48	0,74	1000,03	0,49
g19	1380,76	1,11	1370,42	0,35	1369,78	0,31
g20	1854,17	1,98	1839,67	1,18	1834,43	0,89
Summe	13272,50	1,27	13208,55	0,78	13185,87	0,61

Tabelle 4.20: Lösungen der Multistart-TS_{GPU} auf den Golden-Instanzen

In Tabelle 4.20 finden sich die Ergebnisse des Algorithmus auf den Golden-Instanzen. Auch dieses Verfahren erreicht eine sehr gute Lösungsgüte mit einer Abweichung von unter 0,5% zu den besten bekannten Lösungen. Insgesamt wird die bisher geringste Abweichung erzielt, wobei die Unterschiede zur $VNS \times SimA_{GPU}$ nur marginal sind. Weiterhin weist auch dieses Verfahren die Eigenschaft auf, dass bereits nach einer kurzen Rechenzeit von 2 Minuten eine Abweichung von den besten bekannten Lösungen von unter 1,3% erzielt werden kann. Auch bei Betrachtung der anderen Instanzen (vgl. Tabellen B.22 - B.24) werden keine Überraschungen ersichtlich. Im Rahmen der Gehring-Instanzen werden 11 neue beste Lösungen gefunden. Weiterhin ist die mittlere Abweichung zu den besten bekannten Lösungen mit 0,45% auf ähnlichem Niveau wie die anderen Verfahren.

4.2.2.4 Betrachtung der GAs

Im Rahmen des GA werden die zuvor vorgestellten Verfahren, die einen Großteil der Rechenzeit auf der GPU verbringen, als lokale Suche im Kontext des Mutationsoperators eingesetzt. Es wird, wie bei den anderen Verfahren auch, eine Population von Lösungen verwaltet, deren Anzahl der Anzahl

der SMs entspricht. Nach jedem Kernelaufruf wird die beste Lösung, die von der GPU erzeugt wurde, in die Population übernommen und ersetzt dabei zufällig eine alte Lösung, wobei garantiert ist, dass nicht die beste bisher gefundene Lösung ersetzt und somit gelöscht wird. Nach 20 Kernelaufrufen mit je 100 Iterationen auf der GPU[108] wird der Crossover auf den bestehenden Lösungen ausgeführt. Hierbei wird für jeden Crossover eine Kundensequenz zwischen 1 und 11 selektiert, die ausgetauscht wird. Die so erzeugten 16 Lösungen werden auf der GPU mittels $VNS \times SimA_{GPU}$ und Derivaten bzw. TS_{GPU} optimiert. Weiterhin wird alle 10 Kernelaufrufe mit der bisher besten gefundenen Lösung weitergearbeitet, d.h., alle 16 Blöcke verarbeiten dann die beste gefundene Lösung. Ausnahme bildet die TS_{GPU} bei der jeder Block für sich entscheidet, wann er mit der besten Lösung weiterarbeitet. Auch hier gilt als Abbruchkriterium wieder eine Laufzeit von 3600 Sekunden.

Multistart-$VNS \times SimA_{GPU}$

Die Einbindung der Multistart $VNS \times SimA_{GPU}$ in den GA hat bei Betrachtung der Golden-Instanzen zu einer merklichen Steigerung der Lösungsgüte geführt (siehe Tabelle 4.21). Dabei kann sowohl die mittlere Abweichung der durchschnittlichen Abweichung im Vergleich zur besten gefundenen Lösung auf unter 0,5% reduziert werden, als auch die mittlere Abweichung der besten gefundenen Lösung auf 0,31% gesenkt werden. Weiterhin erreicht das Verfahren bereits nach 2 Minuten eine Lösungsgüte, die weniger als 1% von der besten bekannten Lösung entfernt ist. Die Lösungsgüte kann nach 10 Minuten weiter gesteigert werden und liegt dann bereits bei unter 0,5%. Die merkliche Steigerung der Lösungsgüte im Zusammenhang mit dem GA kann dadurch erklärt werden, dass durch ihn eine weitere Diversifizierungsmethode eingeführt wird. Das heißt, die $VNS \times SimA_{GPU}$ enthält bereits eine „kleine“ Diversifizierungsmethode, die auf einer lokalen Suche aufbaut. Der Crossover führt dazu, dass die vorliegenden Lösungen stark verändert werden und so neue Bereiche des Lösungsraumes erreicht werden können, die mit dem Shaking-Verfahren allein nicht zu erreichen sind.

Ein ähnliches Bild zeichnet sich bei der Betrachtung der Christofides-, Taillard- und Gehring-Instanzen ab (vgl. Tabellen B.25- B.27). Auch hier kann der Algorithmus überzeugen und erzielt in den Gehring-Instanzen mit 0,15% das bisher beste Ergebnis. Allerdings werden von ihm an dieser Stelle „nur“ 10 neue beste Lösungen erzielt. Die Christofides-Instanzen löst das Verfahren mit einer mittleren Abweichung der besten gefundenen Lösungen von 0,04% und die Taillard-Instanzen mit 0,33%.

Aus den Ergebnissen lässt sich folgern, dass durch den Einbezug des Crossovers bzw. die Einbindung der Lokalsuchen in einen GA die bisher besten Resultate erzielt werden können.

Multistart-$VNS \times SimA_{GPU}$ mit adaptiver Abkühlung

Bei Verwendung der $VNS \times SimA_{GPU}$ mit adaptiver Abkühlung im Zusammenhang mit dem GA findet man ein ähnliches Verhalten wie ohne GA. Das heißt, das Endergebnis, das das Verfahren auf den Golden-Instanzen (vgl. Tabelle 4.22) erzielt, liegt auf einem sehr hohen Niveau. Mit einer mittleren Abweichung der besten gefundenen mit den besten bekannten Lösungen von 0,3% übertrifft dieser Algorithmus den GA mit der statischen $VNS \times SimA_{GPU}$. Gleichzeitig ist auch hier auffällig, dass der Algorithmus längere Zeit benötigt, um die guten Resultate zu erreichen. Wo das vorherige Verfahren nach 2 Minuten schon weniger als 1% von den besten bekannten Lösungen entfernt ist, sind es in diesem

[108] Eine Iteration auf der GPU entspricht der Ausführung bzw. der Bewertung eines jeden Moves.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	582,89	581,31	3475,76	0,55	0,28
g10	736,26	740,35	739,49	2847,40	0,56	0,44
g11	912,84	916,99	915,54	3331,79	0,45	0,30
g12	1102,69	1110,20	1107,46	2577,98	0,68	0,43
g13	857,19	858,23	857,19	2756,00	0,12	0,00
g14	1080,55	1081,01	1080,89	1051,70	0,04	0,03
g15	1337,92	1345,04	1342,64	1258,83	0,53	0,35
g16	1612,50	1625,25	1622,24	3132,82	0,79	0,60
g17	707,76	708,38	707,76	3444,79	0,09	0,00
g18	995,13	1000,10	996,92	2167,74	0,50	0,18
g19	1365,60	1370,07	1369,16	3438,79	0,33	0,26
g20	1818,25	1829,63	1826,14	3450,16	0,63	0,43
Summe	13106,40	13168,13	13146,73	32933,77	0,47	0,31

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	584,32	0,80	582,07	0,41	582,07	0,41
g10	742,57	0,86	741,00	0,64	739,86	0,49
g11	920,76	0,87	916,93	0,45	916,67	0,42
g12	1120,76	1,64	1109,91	0,65	1107,75	0,46
g13	861,38	0,49	858,88	0,20	858,38	0,14
g14	1083,46	0,27	1080,97	0,04	1080,89	0,03
g15	1351,70	1,03	1344,65	0,50	1342,64	0,35
g16	1636,41	1,48	1630,01	1,09	1622,35	0,61
g17	708,31	0,08	708,08	0,05	707,86	0,01
g18	1001,80	0,67	998,77	0,37	997,08	0,20
g19	1374,65	0,66	1370,73	0,38	1369,30	0,27
g20	1845,51	1,50	1829,08	0,60	1826,54	0,46
Summe	13231,62	0,96	13171,07	0,49	13151,39	0,34

Tabelle 4.21: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ auf den Golden-Instanzen

Falle noch fast 2,5%. Unter Einbeziehung der anderen Problemklassen zeigen sich keine Überraschungen (vgl. Abbildung B.28 - B.30). Hier liegt die Lösungsgüte auf hohem Niveau, aber sie kann den GA mit Multistart-$VNS \times SimA_{GPU}$ nicht übertreffen. Auch hier scheint sich wieder - gerade bei kleinen Instanzen - die Tatsache bemerkbar zu machen, dass ein Temperaturniveau zu lange gehalten wird, und eine schnellere Abkühlung wahrscheinlich zu besseren Ergebnissen in kürzerer Zeit führen könnte.

Multistart-$VNS \times SimA_{GPU}$ mit COSA

Die Variante des GA, die zusätzlich noch COSA als Shaking-Verfahren verwendet, befindet sich bzgl. der Lösungsgüte und bei Betrachtung der Golden-Instanzen auf dem Niveau des GA mit Multistart-$VNS \times SimA_{GPU}$ (vgl. Tabelle 4.23). Völlig wird das Level der beiden anderen bisher vorgestellten GAs nicht erreicht. Auch hier zeigt sich die Schwäche der Diversifizierung bei stark strukturierten Problemen. Bei den weniger stark strukturierten bzw. zufällig erzeugten Probleminstanzen beweist das Verfahren und mit ihr das Shaking-Verfahren seine Stärke. Dabei gehört es bei den Christofides- und Taillard-Instanzen (siehe Tabelle B.34 und B.32) mit einer mittleren Abweichung der besten gefundenen zu den besten bekannten Lösungen von 0,1% bzw. 0,33% zu den besten in dieser Arbeit implementierten Verfahren. Weiterhin gelingt es dem Algorithmus bei Betrachtung der Gehring-Instanzen 13 neue beste Lösungen zu ermitteln (vgl. Tabelle B.33). Außerdem wird bei diesen Instanzen im Vergleich zu den anderen Verfahren eine sehr geringe mittlere Abweichung von lediglich 0,06% erzielt.

Multistart-TS_{GPU}

Zum Schluss soll die TS als Lokalsuche im Kontext des GA betrachtet werden. Hierbei fällt auf, dass sich die Lösungsgüte bei den Golden-Instanzen bezogen auf die mittlere Abweichung marginal

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	583,93	582,30	3206,30	0,73	0,45
g10	736,26	739,70	738,54	1478,19	0,47	0,31
g11	912,84	917,33	916,80	3518,59	0,49	0,43
g12	1102,69	1109,72	1105,57	3421,53	0,64	0,26
g13	857,19	859,62	857,19	2281,41	0,28	0,00
g14	1080,55	1081,64	1080,83	3129,99	0,10	0,03
g15	1337,92	1345,81	1343,11	2692,13	0,59	0,39
g16	1612,50	1623,75	1620,43	3451,77	0,70	0,49
g17	707,76	708,60	708,02	3256,87	0,12	0,04
g18	995,13	1000,15	999,39	3275,67	0,50	0,43
g19	1365,60	1368,82	1367,34	3130,44	0,24	0,13
g20	1818,25	1828,47	1826,02	3111,33	0,56	0,43
Summe	13106,40	13167,54	13145,54	35954,21	0,47	0,30

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	594,71	2,59	583,42	0,64	583,05	0,58
g10	748,96	1,72	744,03	1,06	738,54	0,31
g11	934,28	2,35	925,55	1,39	917,65	0,53
g12	1136,85	3,10	1133,75	2,82	1118,57	1,44
g13	876,88	2,30	858,93	0,20	858,28	0,13
g14	1111,32	2,85	1088,74	0,76	1080,89	0,03
g15	1379,16	3,08	1367,38	2,20	1345,31	0,55
g16	1665,90	3,31	1658,96	2,88	1635,85	1,45
g17	712,88	0,72	709,63	0,26	708,06	0,04
g18	1015,25	2,02	1009,56	1,45	999,84	0,47
g19	1381,98	1,20	1377,59	0,88	1367,78	0,16
g20	1870,81	2,89	1850,95	1,80	1834,55	0,90
Summe	13428,99	2,46	13308,49	1,54	13188,37	0,63

Tabelle 4.22: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Golden-Instanzen

verschlechtert hat im Vergleich zur TS_{GPU} ohne den GA (vgl. Tabelle 4.24). Allerdings ist die Verschlechterung zu gering, um aussagekräftige Schlussfolgerungen daraus ziehen zu können. Außerdem erreicht auch dieser Algorithmus bereits nach kurzer Zeit sehr gute Lösungsqualitäten. Im Kontext der Christofides- und Taillard-Instanzen (Tabellen B.34 und B.35) fällt das sehr hohe Niveau der Lösungsgüte auf, bei dem das Verfahren mit einer mittleren Abweichung von 0,04% sowie 0,29% sehr gut abschneidet. Die Gehring-Instanzen löst das Verfahren ebenfalls sehr gut und erzielt wie der GA mit $VNS \times SimA_{GPU}$ und COSA 13 neue beste Lösungen (vgl. Tabelle B.36).

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	583,90	582,35	1206,31	0,72	0,45
g10	736,26	740,03	739,11	1862,21	0,51	0,39
g11	912,84	916,26	914,56	3027,04	0,38	0,19
g12	1102,69	1110,84	1108,07	2963,53	0,74	0,49
g13	857,19	860,78	859,65	669,61	0,42	0,29
g14	1080,55	1080,83	1080,55	1220,93	0,03	0,00
g15	1337,92	1346,77	1345,25	3310,60	0,66	0,55
g16	1612,50	1626,52	1624,74	1362,39	0,87	0,76
g17	707,76	708,41	708,02	3512,32	0,09	0,04
g18	995,13	1001,74	1000,07	2325,29	0,66	0,50
g19	1365,60	1371,95	1368,22	3192,60	0,47	0,19
g20	1818,25	1831,12	1825,59	2618,65	0,71	0,40
Summe	13106,40	13179,15	13156,16	27271,48	0,56	0,38

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	585,30	0,96	584,07	0,75	582,35	0,45
g10	743,38	0,97	741,42	0,70	739,11	0,39
g11	921,29	0,93	916,03	0,35	915,61	0,30
g12	1122,55	1,80	1112,03	0,85	1109,48	0,62
g13	862,35	0,60	859,65	0,29	859,65	0,29
g14	1084,68	0,38	1080,89	0,03	1080,55	0,00
g15	1353,08	1,13	1348,62	0,80	1345,77	0,59
g16	1638,01	1,58	1628,27	0,98	1624,74	0,76
g17	709,11	0,19	708,71	0,13	708,40	0,09
g18	1008,52	1,35	1003,73	0,86	1000,13	0,50
g19	1379,35	1,01	1373,42	0,57	1371,32	0,42
g20	1845,07	1,48	1829,27	0,61	1826,31	0,44
Summe	13252,70	1,12	13186,12	0,61	13163,42	0,44

Tabelle 4.23: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Golden-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
g9	579,71	583,63	582,94	3052,16	0,68	0,56
g10	736,26	742,26	740,94	2658,65	0,82	0,64
g11	912,84	918,83	918,03	3123,59	0,66	0,57
g12	1102,69	1115,87	1112,53	3516,74	1,20	0,89
g13	857,19	860,76	859,46	2422,21	0,42	0,26
g14	1080,55	1081,13	1080,89	2702,27	0,05	0,03
g15	1337,92	1346,44	1343,18	2683,45	0,64	0,39
g16	1612,50	1628,70	1625,51	3127,24	1,00	0,81
g17	707,76	708,20	708,03	2388,32	0,06	0,04
g18	995,13	999,86	996,80	3541,21	0,47	0,17
g19	1365,60	1372,19	1369,47	2597,87	0,48	0,28
g20	1818,25	1837,26	1834,70	2455,68	1,05	0,90
Summe	13106,40	13195,13	13172,49	34269,39	0,68	0,50

Instanz	Beste Lösung (2 Min)	Abweichung [%]	Beste Lösung (10 Min)	Abweichung [%]	Beste Lösung (30 Min)	Abweichung [%]
g9	586,27	1,13	583,89	0,72	583,36	0,63
g10	745,42	1,24	743,06	0,92	742,16	0,80
g11	923,36	1,15	920,77	0,87	918,15	0,58
g12	1121,55	1,71	1117,95	1,38	1113,63	0,99
g13	860,76	0,42	859,46	0,26	859,46	0,26
g14	1084,41	0,36	1081,60	0,10	1080,89	0,03
g15	1355,21	1,29	1350,60	0,95	1347,57	0,72
g16	1640,51	1,74	1631,55	1,18	1625,58	0,81
g17	709,43	0,24	708,22	0,07	708,06	0,04
g18	1004,86	0,98	1000,01	0,49	997,62	0,25
g19	1379,16	0,99	1370,62	0,37	1369,47	0,28
g20	1856,70	2,11	1842,78	1,35	1838,36	1,11
Summe	13267,64	1,23	13210,50	0,79	13184,32	0,59

Tabelle 4.24: Lösungen des GA mit Multistart-TS_{GPU} auf den Golden-Instanzen

4.2.3 Zusammenfassung

Nachdem sämtliche Metaheuristiken, die in der Arbeit verwendet werden, vorgestellt wurden, soll nun eine kurze Zusammenfassung gegeben sowie eine direkte Positionierung der Algorithmen untereinander vorgenommen werden. Überdies wird nach der direkten Gegenüberstellung der Algorithmen des Beitrags eine Gegenüberstellung der Verfahren mit denen der Literatur vorgenommen.

4.2.3.1 Positionierung der implementierten Algorithmen untereinander

Zur besseren Lesbarkeit der folgenden Tabellen und Abbildungen werden die hier vorgestellten Verfahren nach Tabelle 4.25 abgekürzt.

Verfahren	Abkürzung
$VNS \times SimA_{CPU}$	VSC
TS_{CPU}	TC
$VNS \times SimA_{GPU}$	VSG
$VNS \times SimA_{GPU}$ mit adaptiver Abkühlung	VSGA
$VNS \times SimA_{GPU}$ mit COSA	VSGC
TS_{GPU}	TG
GA mit $VNS \times SimA_{GPU}$	GVSG
GA mit $VNS \times SimA_{GPU}$ mit adaptiver Abkühlung	GVSGA
GA mit $VNS \times SimA_{GPU}$ mit COSA	GVSGC
GA mit TS_{GPU}	GTG

Tabelle 4.25: Abkürzungen der implementierten Metaheuristiken

In Tabelle 4.26 sind alle Verfahren mit ihren jeweils besten gefundenen Lösungen und ihrer daraus resultierenden mittleren Abweichung zu den besten bekannten Lösungen im Kontext der Golden-Instanzen dargestellt. Hieraus lässt sich ablesen, dass der GVSGA das beste Resultat hinsichtlich der Lösungsgüte erzielt. Vom Versuchsaufbau her sind die Laufzeiten bzw. das Abbruchkriterium der Verfahren mit 3600 Sekunden identisch. Die Zeiten, die hier und auch in den anderen bereits dargestellten Tabellen abgedruckt sind, stellen die Zeiten bzw. die kumulierten Zeiten dar, wann die beste gefundene Lösung ermittelt wurde. Die Ergebnisse verdeutlichen, dass bereits die einfache Nutzung der Operatoren, wie sie in Abschnitt 4.1 vorgestellt werden, sehr gute Ergebnisse erzielen. Da die Ressourcen der GPU dadurch nicht vollständig genutzt werden, ist es möglich, weitere Aufgabenteile auf die GPU auszulagern, bzw. die Grafikkarte mit zusätzlichen Aufgaben zu belasten. Durch diese zusätzliche Belastung in Form von mehreren Lösungen, die parallel bearbeitet werden, kann die Lösungsqualität weiter gesteigert werden. In diesem Zusammenhang wird deutlich, dass der VSGC am wenigsten gute Resultate erzielt, was eventuell durch die strukturierten Probleme, die bei den Golden-Instanzen vorhanden sind, erklärt werden kann. Eine weitere erhebliche Steigerung der Lösungsgüte kann durch die Integration der Lokalsuchen in einen GA erzielt werden. Auch hierbei wird ersichtlich, dass die Lokalsuchen, die naturanaloge Elemente innehaben, bessere Ergebnisse erzielen als die rein künstliche TS.

Tabelle 4.26: Beste Lösungen der implementierten Metaheuristiken für die Golden-Instanzen

Instanz	Beste bekannte	VSC	TC	VSG	VSGA	VSGC	TG	GVSG	GVSGA	GVSGC	GTG
g9	579,71	583,97	582,35	582,82	582,41	580,78	583,48	581,31	582,30	582,35	582,94
g10	736,26	741,68	739,00	741,02	739,96	740,83	741,06	739,49	738,54	739,11	740,94
g11	912,84	920,77	916,01	918,19	917,32	918,04	915,62	915,54	916,80	914,56	918,03
g12	1102,69	1110,58	1110,50	1110,22	1112,23	1109,03	1112,94	1107,46	1105,57	1108,07	1112,53
g13	857,19	859,59	859,55	859,53	859,19	859,61	859,65	857,19	857,19	859,65	859,46
g14	1080,55	1080,55	1080,89	1080,90	1080,90	1081,96	1081,02	1080,89	1080,83	1080,55	1080,89
g15	1337,92	1347,58	1344,51	1345,42	1345,78	1345,33	1344,70	1342,64	1343,11	1345,25	1343,18
g16	1612,50	1624,22	1618,46	1622,67	1624,07	1625,41	1623,17	1622,24	1620,43	1624,74	1625,51
g17	707,76	707,80	707,91	707,90	707,93	708,60	708,22	707,76	708,02	708,02	708,03
g18	995,13	1001,14	997,96	1001,71	1001,29	1003,14	1000,03	996,92	999,39	1000,07	996,80
g19	1365,60	1369,15	1375,70	1369,17	1370,77	1371,41	1369,56	1369,16	1367,34	1368,22	1369,47
g20	1818,25	1836,42	1849,45	1828,91	1829,13	1835,77	1829,94	1826,14	1826,02	1825,59	1834,70
Summe	13106,40	13183,46	13182,29	13168,44	13170,96	13179,91	13169,41	13146,73	13145,54	13156,16	13172,49
Zeit [s]		34039,89	31324,46	35134,91	37365,01	33712,42	33223,49	32933,77	35954,21	27271,48	34269,39
Abweichung [%]	0,00	0,59	0,58	0,47	0,49	0,56	0,48	0,31	0,30	0,38	0,50

Bisher wird lediglich die Lösungsqualität als Kriterium zur Beurteilung der Heuristiken herangezogen. Nun soll zusätzlich noch die Laufzeit betrachtet werden, wobei hier auf den Zeitpunkt des Findens der besten Lösung im Suchprozess abgestellt wird. In Abbildung 4.22 ist in x-Richtung die kumulierte Zeit in Sekunden abgetragen und in y-Richtung die kumulierten Distanzen der besten gefundenen Lösungen. Im konkret betrachteten Fall zeigt sich, dass die Verfahren GVSGC, GVSG sowie GVSGA die anderen Verfahren dominieren. Das heißt, GVSGC dominiert alle Verfahren bis auf GVSG und GVSGA. Hierbei kann der Algorithmus des GVSGC die schlechtere Lösungsgüte im Vergleich zu den anderen Verfahren durch eine schnellere Ausführungszeit wettmachen.

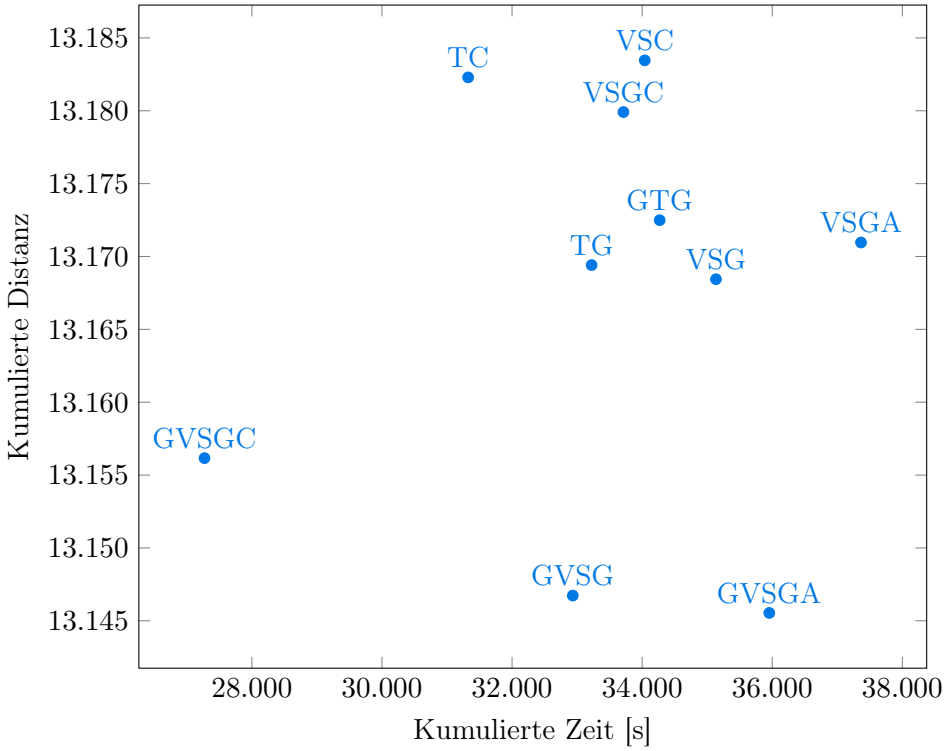


Abbildung 4.22: Positionierung der Metaheuristiken (Golden-Instanzen)

Da für die Gehring-Instanzen viele neue Lösungen im Rahmen dieser Arbeit gefunden werden, sollen diese nochmals gesondert betrachtet werden (vgl. Tabelle 4.27).[109] Insgesamt können die hier vorgestellten Metaheuristiken 16 neue beste Lösungen auf den Gehring-Instanzen ermitteln, was ihre Stärke bzgl. der Lösungsqualität unterstreicht. Die neuen besten Lösungen sind im Anhang A.3 zu finden. In Abbildung 4.23 sind die kumulierten Werte der einzelnen Metaheuristiken dargestellt. Auch hier zeigt sich, dass gerade die Verwendung der naturanalogen und GPU-auslastenden Verfahren nicht nur die höchste Lösungsgüte liefern (GVSGC), sondern auch recht schnell zu guten Ergebnissen kommen. Was auffällt ist, dass das künstliche Verfahren der TS auf allen Ebenen zwar gute Ergebnisse erzielt, jedoch die Dauer bis zum Erreichen dieser Ergebnisse die der anderen Verfahren oftmals übersteigt.

Die bisherigen Ergebnisse lassen die Schlussfolgerung zu, dass man mit möglichst optimaler Ausnutzung der Hardware bzw. mit Verfahren, die parallel mehrere Lösungen bearbeiten, sehr viel bessere Lösungen berechnen kann als im anderen Fall. Hierbei wiegt das Weniger an Iterationen pro Lösung

[109] Die zusammengefassten Ergebnistabellen für die Christofides- und Taillard-Instanzen finden sich im Anhang in den Tabellen B.37 und B.38.

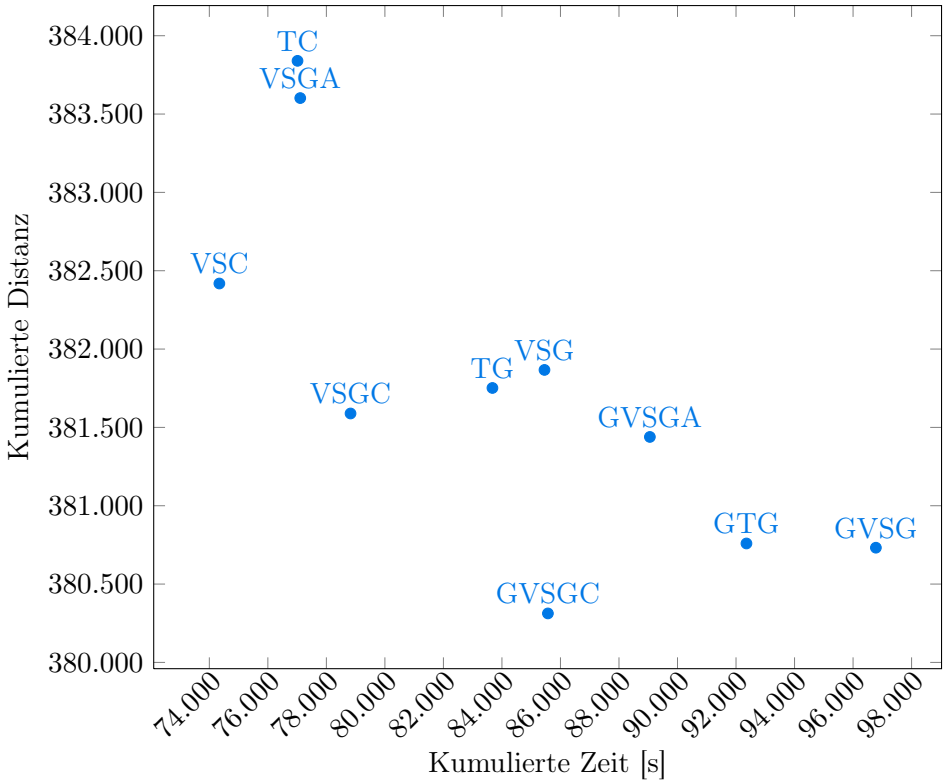


Abbildung 4.23: Positionierung der Metaheuristiken (Gehring-Instanzen)

weniger schwer als das Mehr an Lösungen und damit einhergehend das Mehr beim Absuchen des Lösungsraumes. Weiterhin spielt die Art der Diversifizierung eine große Rolle. Wenn man annimmt, dass der GA bzw. der Crossover, der im GA ausgeführt wird, als ein großer Diversifizierungsschritt angesehen wird, so unterstreichen die Ergebnisse die Bedeutung der Diversifizierung. Gleiches gilt für das Shaking-Verfahren. Auch hier kann durch die Art der Diversifizierung großer Einfluss auf die Resultate genommen werden.

Die von Barr et al. (1995) vorgeschlagenen Kriterien von Metaheuristiken (vgl. Abschnitt 2.5.1) werden bis auf den Tradeoff zwischen Gültigkeit und Lösungsgüte angegeben. Es kann gezeigt werden, dass die Algorithmen bereits nach sehr kurzer Zeit gute Lösungen erreichen können. Weiterhin kann auf die Robustheit des Algorithmus durch Betrachtung der durchschnittlichen Abweichung der fünf Testläufe geschlossen werden. Auch hier zeigt sich im Mittel eine geringe Abweichung aller Verfahren im Vergleich zu den besten bekannten Lösungen. Außerdem zeigt sich, dass die Verfahren auch auf unterschiedlichen Benchmarks sehr gute Ergebnisse erzielen, was ein weiteres Zeichen für die Robustheit der Verfahren ist.

Instanz	Beste bekannte	VSC	TC	VSG	VSGA	VSGC	TG	GVSG	GVSGA	GVSGC	GTG
C1_200	2570,49	2567,68	**2567,67**	**2567,67**	**2567,67**	**2567,67**	**2567,67**	**2567,67**	**2567,67**	**2567,67**	**2567,67**
C1_400	6765,03	6735,41	6734,11	6728,92	6738,28	6731,46	6721,28	6725,28	6730,88	6730,04	**6721,10**
C1_600	13465,24	13462,80	13452,24	13449,25	13498,26	13450,30	13454,98	13440,16	13476,80	**13436,00**	13437,92
C1_800	23999,35	**23690,76**	23744,66	23726,37	23795,78	23728,48	23735,78	23705,29	23767,77	23713,51	23712,53
C1_1000	39811,15	39699,29	39609,58	39455,07	40035,36	39590,23	39567,37	39441,30	39662,48	39379,69	**39364,68**
C2_200	-	1484,34	1488,38	1483,59	1483,00	**1482,43**	**1482,43**	**1482,43**	**1482,43**	**1482,43**	**1482,43**
C2_400	-	3130,58	3173,43	3132,58	3127,41	3125,40	3130,99	3124,63	3125,99	3124,22	**3123,99**
C2_600	-	6323,97	6301,32	6318,06	6316,76	6298,84	6284,85	6304,96	6303,97	**6269,95**	6280,19
C2_800	-	9719,78	9668,65	9658,25	9681,31	9662,57	9626,61	9612,74	9607,58	**9601,51**	9651,14
C2_1000	-	14491,25	14405,01	14320,04	14308,41	14354,37	14293,59	14317,84	14289,51	**14258,22**	14287,64
R1_200	2878,24	2887,31	2879,01	2872,02	2874,99	2870,49	2869,81	**2869,66**	2871,26	2871,26	2871,44
R1_400	7192,03	7238,46	7193,39	7186,31	7204,41	7186,01	7176,58	7187,14	7193,66	7184,40	**7176,31**
R1_600	15691,37	15700,29	15625,16	15618,17	15679,00	15617,23	**15517,54**	15573,38	15664,68	15562,22	15525,05
R1_800	27650,67	27983,38	27784,29	27897,53	28316,22	27959,21	27772,71	27775,81	27972,24	27880,56	27682,63
R1_1000	42311,91	42974,66	43101,07	42936,90	43814,41	42934,79	42765,54	42832,43	43202,86	42737,05	42811,79
R2_200	1610,30	1611,24	1625,51	1610,30	1610,30	1610,30	1610,30	1611,74	1616,48	1610,30	1610,30
R2_400	3323,05	3346,68	3352,50	3373,99	3312,88	3347,29	3318,24	3343,06	3319,54	3340,40	**3312,87**
R2_600	6254,17	6295,97	6299,72	6288,14	6229,18	6218,15	6282,18	6306,63	6233,44	**6188,39**	6267,77
R2_800	10230,06	10319,47	10389,16	10463,03	10261,44	10281,49	10337,10	10388,83	10232,92	10313,08	10324,43
R2_1000	15149,40	15202,40	15658,62	15317,36	15154,15	15217,47	15530,79	15249,71	**15090,29**	15112,32	15344,56
RC1_200	2804,20	2817,15	2825,54	2803,34	2814,01	2809,68	2804,95	**2800,78**	2803,36	2813,63	2800,78
RC1_400	7381,47	7331,79	7360,85	7311,44	7271,66	7294,71	7278,29	7273,88	**7267,21**	7299,03	7269,95
RC1_600	14786,15	14889,15	15006,56	14854,92	14925,46	14873,83	14871,57	14833,25	14836,39	14838,30	14801,89
RC1_800	26720,23	26865,81	27025,62	26836,10	27022,57	26819,72	26770,90	26760,54	26889,81	26821,16	**26655,82**
RC1_1000	41583,54	42023,63	42128,18	41942,25	42152,89	42015,10	41936,29	41717,23	41848,51	41780,51	41910,91
RC2_200	1513,00	1513,00	1513,00	1514,74	1514,71	1513,00	1513,00	1513,00	1513,00	1513,00	1513,00
RC2_400	3100,06	3111,64	3116,26	3123,10	3103,82	3100,86	**3090,19**	3106,24	3106,20	3094,14	3097,45
RC2_600	5732,08	5802,39	5882,77	5820,80	5751,70	5723,67	5766,66	5739,38	5763,01	**5715,92**	5766,39
RC2_800	9217,99	9318,72	9714,06	9417,12	9355,51	9379,29	9433,13	9406,53	9329,75	9369,43	9372,70
RC2_1000	13631,76	13879,33	14213,28	13839,47	13680,40	13824,70	14240,32	13720,54	13669,40	13704,25	14013,79
Summe	-	382418,33	383839,60	381866,83	383601,97	381588,73	381751,63	380732,08	381439,07	380312,60	380759,10
Zeit [s]		74342,73	77013,84	85453,98	77107,01	78825,65	83675,13	96780,11	89055,98	85571,11	92356,56
Abweichung [%]	0,00	0,55	0,99	0,46	0,96	0,37	0,45	0,15	0,36	0,06	0,16

Tabelle 4.27: Beste Lösungen der implementierten Metaheuristiken für die Gehring-Instanzen

4.2.3.2 Positionierung der implementierten Algorithmen im Vergleich zu Lösungsverfahren der Literatur

Bei dem Vergleich mit anderen Lösungsverfahren sollen nur diese der Literatur betrachtet werden, die eine Abweichung von unter 2% zu den besten bekannten Lösungen im Rahmen der Golden-Instanzen erreichen. In Tabelle 4.28 finden sich die Verfahren mit der erreichten Lösungsgüte, der benötigten Zeit sowie der erzielten Abweichung. Um diese Heuristiken mit den GPU-Verfahren auch auf der Zeitebene vergleichen zu können, ist es notwendig, die Zeiten, die bei Berechnung mit der GPU ermittelt werden, anzupassen. Dem Autor sind keine Veröffentlichungen zum Umrechnen von GPU-Zeiten auf CPU-Zeiten bekannt. Deshalb soll hier auf den Speedup-Gedanken zurückgegriffen werden. Im Rahmen des Speedups geht es weniger darum, die Zeiten anzugleichen, sondern sehr viel mehr darum herauszufinden, welche Leistungssteigerung man durch die zusätzliche Hardware erreichen kann. Das heißt, in diesem Beitrag kommt ein Intel Core i7 860 Prozessor zum Einsatz, der bei der Berechnung der Zeiten in dieser Arbeit berücksichtigt werden muss, da auch dieser Arbeit übernimmt. Der Prozessor erzielt mit dem Linpack-Benchmark, mit dem auch die MFlop/s der anderen Prozessoren, die in der Literatur verwendet werden, bestimmt werden, 2503 MFlop/s. Daraus ergibt sich, dass die Zeiten dieser Arbeit mit einem Zeitfaktor von $\frac{2503}{1779} \approx 1,41$ multipliziert werden.[110] Tabelle 4.28 enthält weiterhin die besten Lösungen der hier gezeigten Verfahren, wobei die Zeiten, wie beschrieben, angepasst sind. Weiterhin wird noch das VSG-Verfahren mit nur 2 Minuten Laufzeit aufgenommen.

Aus der Tabelle resultiert Abbildung 4.24 in die der erreichte Zielfunktionswert der Verfahren in Abhängigkeit von Rechendauer abgetragen ist.[111] Die Verfahren der Literatur sind in blau dargestellt, die in der Arbeit implementierten andersfarbig. In der Abbildung werden nur die besten Verfahren dieser Arbeit im Sinne der Lösungsgüte dargestellt. Hierbei zeigt sich, dass sich die hier umgesetzten GAs unter den Top6-Verfahren befinden. Dies ist auch einhergehend mit der Tendenz, dass GAs im Allgemeinen sehr gute Resultate im Kontext der Tourenplanung erzielen. Allerdings setzen Autoren wie Nagata und Bräysy (2009) oder Vidal et al. (2011) sehr viel komplexere Mechanismen ein, um einen Crossover durchzuführen. Das wiederum spricht für die Einfachheit der hier vorgestellten Algorithmen. Gleiches gilt bei Betrachtung des Algorithmus von Groër et al. (2011), bei dem neben der Implementierung einer Metaheuristik auch die Implementierung eines exakten Verfahrens vonnöten ist. Das heißt, bei der Wahl des Verfahrens muss zwischen der Einfachheit der Implementierung sowie der Lösungsgüte, die erreicht werden soll, unterschieden werden. Dabei darf natürlich nicht verschwiegen werden, dass auch das hier umgesetzte Verfahren durch Verwendung der GPU Schwierigkeiten in der Implementierung mit sich bringt. Dennoch sollte es für einen erfahrenen Entwickler nicht schwer sein, die Algorithmen, wie beschrieben, umzusetzen. Wenn die Hürde der GPU-Programmierung überwunden ist, ist es möglich, einfachste Operatoren zum Kanten- bzw. Knotenaustausch zu verwenden. Ebenso kann auf komplexe Crossover-Verfahren verzichtet werden. Allerdings zeigt sich auch, dass die Verfahren bzgl. der

[110] Hier muss man darauf hinweisen, dass nur der Zeitanteil der auch tatsächlich auf der CPU berechnet wird, berücksichtigt werden sollte. Das heißt, theoretisch ergibt sich die Gesamtrechenzeit aus CPU-Zeit multipliziert mit dem Zeitfaktor zuzüglich der GPU-Zeit. Durch die Multiplikation der Gesamtzeit mit dem Zeitfaktor resultiert tendenziell eine Zeitüberschätzung statt eine -unterschätzung, wodurch sich die Ergebnisse weiter positiv zur GPU hin bewegen sollten.

[111] Die Abkürzungen in der Abbildung finden sich in Tabelle B.14 wieder. Weiterhin wird auch in dieser Grafik auf das Verfahren JCLb verzichtet, da das Verfahren die Grafik zu sehr verzerrt. Auch die Verfahren GGWa und T werden der Übersichtlichkeit halber nicht dargestellt.

Verfahren	Zielfunktionswert	Zeit	Abweichung
Alabas-Uslu und Dengiz (2011)	13314,27	13739,11	1,59
Chen et al. (2010)	13257,86	264,38	1,16
Cordeau und Maischberger (2012)	13210,68	18467,28	0,80
Ergun et al. (2006)	13298,12	9274,87	1,46
Groër et al. (2010)	13310,26	739,65	1,56
Groër et al. (2011)	13109,43	140016,86	0,02
Jin et al. (2011)	13124,38	2725849,58	0,14
Jin et al. (2012)	13173,29	46582,15	0,51
Li et al. (2005)	13307,22	127,97	1,53
Lin et al. (2009)	13322,69	77850,22	1,65
Marinakis (2012)	13353,57	171,27	1,89
Marinakis und Marinaki (2010)	13238,05	744,01	1,00
Mester und Bräysy (2007)	13152,40	2394,14	0,35
Nagata und Bräysy (2008)	13124,91	4801,39	0,14
Nagata und Bräysy (2009)	13130,36	16221,43	0,18
Prins (2009)	13213,68	4996,92	0,82
Reimann et al. (2004)	13329,12	4181,79	1,70
Tarantilis (2005)	13249,61	1525,60	1,09
Vidal et al. (2011)	13106,47	57108,10	0,00
VSC	13183,46	47996,24	0,59
TC	13182,29	44167,49	0,58
VSG	13168,44	49540,22	0,47
VSGA	13170,96	52684,66	0,49
VSGC	13179,91	47534,51	0,56
TG	13169,41	46845,12	0,48
GVSG	13146,73	46436,62	0,31
GVSGA	13145,54	50695,44	0,30
GVSGC	13156,16	38452,78	0,38
GTG	13172,49	48319,84	0,50
VSG (2 Minuten)	13231,62	2030,40	0,96

Tabelle 4.28: Vergleich der besten Lösungsverfahren der Literatur mit den implementierten GPU-Verfahren

Rechendauer im Vergleich zu den meisten anderen Verfahren sehr viel Zeit in Anspruch nehmen. Der Grund dafür liegt darin, dass herausgefunden werden soll, welche Lösungsgüte mit einfachen Mitteln und durch Unterstützung der GPU zu erreichen ist. Gleichzeitig wird jedoch ebenfalls ersichtlich, dass das VSG-Verfahren bereits nach 2 Minuten viele der Verfahren der Literatur übertreffen kann - sowohl was die Ausführungsdauer als auch die Lösungsgüte anbelangt.

4.3 Betriebswirtschaftlicher Vergleich

Eine wichtige Frage, die sich bei der Parallelisierung der Algorithmen stellt, ist die der Wirtschaftlichkeit. Hier muss man entscheiden, ob sich der Ressourceneinsatz in Form von Personal durch Parallelisierung von Code auf der GPU rechnet oder ob man eine längere Rechendauer akzeptiert und dafür den Einsatz von Entwicklungskosten spart.

Allerdings soll an dieser Stelle weniger auf die Entwicklungskosten der Software eingegangen werden[112], sondern vielmehr auf die Unterschiede zwischen einer parallelen reinen CPU-Version und dem Pendant der $VNS \times Sim A_{GPU}$. Die parallele CPU-Version wird mittels OpenMP umgesetzt, die die Sprache C/C++ um einfache Direktiven erweitert, um Programme zu parallelisieren. Hierbei sind die Einstiegshürden etwas niedriger als bei der Programmierung mit CUDA, da sich bspw. nicht zwangsläufig mit unterschiedlichen Speicherarten beschäftigt werden muss.

[112] Grundsätzlich wird der Aufwand zur Implementierung eines parallelen Algorithmus von Kim und Bond (2009, Abb. 12) mittels OpenMP als geringer eingeschätzt als der bei Verwendung von CUDA. Allerdings ist aus Sichtweise der Programmiersprache C, d.h., wenn C als Referenz angesehen wird, der Aufwand zum Erlernen von CUDA nach den Autoren nicht sonderlich hoch anzusiedeln.

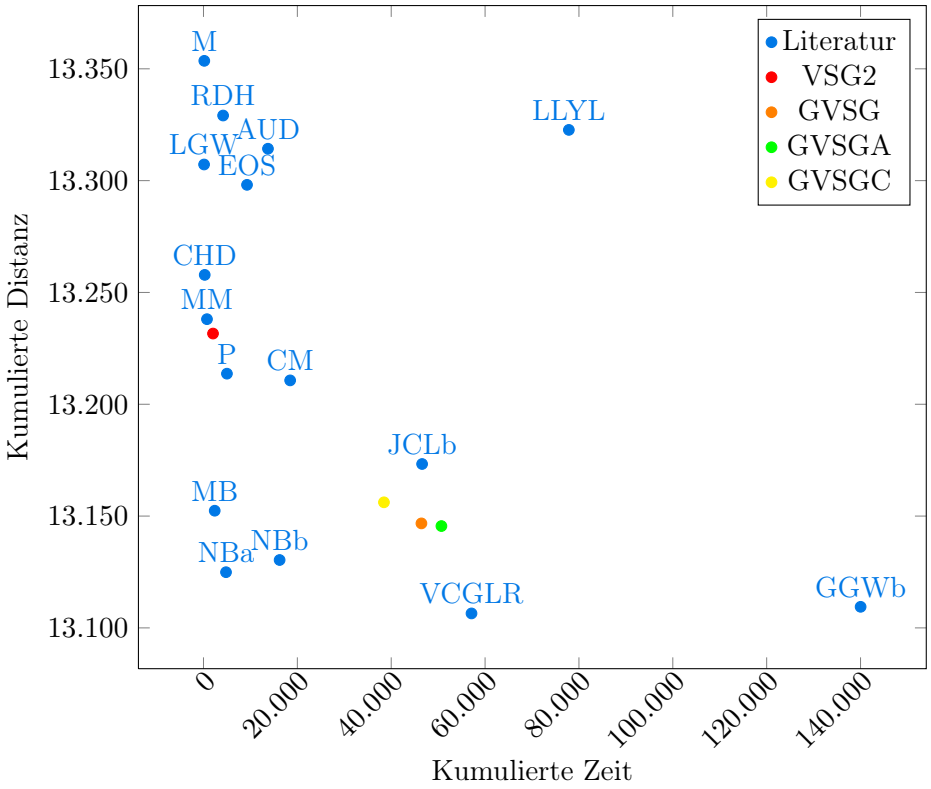


Abbildung 4.24: Positionierung gegenüber Lösungsverfahren der Literatur

In der CPU-Version der $VNS \times SimA_{GPU}$ entspricht die Anzahl der Threads, die gestartet werden, der Populationsgröße. Hier zeigt sich, dass diese Version grob granularer ist als die $VNS \times SimA_{GPU}$, da bei dieser mehr Threads gestartet werden, die Teile der Arbeit übernehmen; also ein Thread bearbeitet in Analogie zu dem Blockkonzept der GPU eine Lösung. Allerdings stehen diesem Thread auf der CPU keine weiteren Threads zur Aufgabenerledigung zur Verfügung. Er muss sich vollständig um die Bewertung der Nachbarschaften bzw. Bestimmung des besten Moves, die in der $VNS \times SimA_{GPU}$ weiter parallelisiert werden können, kümmern. Trotzdem ermöglicht es dieser Aufbau, dass viele Kerne (= entsprechend der Anzahl der Lösungen) an der Lösung des CVRPs beteiligt werden können.

Der folgende Wirtschaftlichkeitsvergleich basiert auf den Kosten, die anfallen, wenn man bei Amazon.com Rechenkapazitäten mietet. Die Rechenzeiten können individuell und stundenweise eingestellt werden. Es gibt zahlreiche Konfigurationen von Rechnern bzw. Kapazitäten, die über den Dienst Amazon Elastic Cloud Computing (EC2) angeboten werden (vgl. Amazon.com (2012b)). Je nach gewünschter Rechenleistung ergibt sich ein entsprechender Mietpreis (vgl. Amazon.com (2012a)).[113]

Angeboten werden dabei auch Rechner, die GPUs enthalten. Bei diesen handelt es sich um eine Tesla M2050 von Nvidia, die ebenfalls auf der Fermi-Architektur aufbaut. Damit ergeben sich nur geringe Unterschiede zur in dieser Arbeit verwendeten GPU. Deshalb werden die Berechnungen, die bereits auf der GTX 580 in dieser Arbeit durchgeführt werden, nicht mehr in der Cloud ausgeführt, sondern es soll ein Mapping der Zeiten vorgenommen werden. Das heißt, es soll die Rechenzeit bestimmt werden, die die Tesla M2050 zur Berechnung der Instanzen, die auf der GTX 580 durchgeführt werden, benötigt.

[113] Es gibt verschiedene Arten, wie man die Instanzen mieten kann. In dieser Arbeit wird von den sog. On-Demand Instances ausgegangen, die sofort nach Registrierung verwendet werden können.

Als Grundlage und aufgrund der wenigen Unterschiede der beiden Hardwarearchitekturen soll auf die theoretisch möglichen, maximalen GFlop/s zurückgegriffen werden. Die Tesla M2050 besitzt eine theoretische Leistung von 1030 GFlop/s (Nvidia, 2011c), wohingegen die GTX 580 eine theoretische Leistung von 1581 GFlop/s (Andermahr, 2010) erbringen kann. Daraus ergibt sich ein Skalierungsfaktor der Ausführungsdauer der bisherigen Testdurchläufe auf der GPU von $\frac{1581}{1030} \approx 1,53$.[114]

Zur Ermittlung des CPU-Pendants wird die Rechenleistung von Amazon.com angemietet. Es wird hierbei auf die Cluster Compute Instance Quadruple Extra Large (API-Name cc1.4xlarge) zurückgegriffen, die pro Stunde 1,24 Euro[115] kostet. Diese Konfiguration besitzt 23 GByte Arbeitsspeicher sowie 2 Intel Xeon X5570 Quad-Core (Nehalem-Architektur). Pro Prozessor stehen 4 Kerne, d.h. insgesamt 8 zur Verfügung, die durch die Hyperthreading-Technologie von Intel auf 16 „erhöht“[116] werden. Somit muss bei paralleler Berechnung von 16 Lösungen ein Kern jeweils 2 Lösungen betrachten. Die Referenz GPU-Instanz besteht aus der gleichen Konfiguration wie der ebenen beschriebenen, mit dem Unterschied der zwei Tesla-Karten, die zur Verfügung stehen. Der Preis pro Stunde Rechenzeit beträgt für diese 2 Euro.

Nun soll in Anlehnung an Wendt (1995, S. 185ff.) die Wirtschaftlichkeit der GPU-Version im Vergleich zur auf dem Amazon-Cluster ausgeführten CPU-Version betrachtet werden. Dabei werden die Distanz, die als Lösung geliefert wird, sowie die Zeit, die zur Berechnung dieser Distanz benötigt wird, als Produktionsfaktoren angesehen.[117] In Abbildungen 4.25, 4.26 und 4.27 sind die Lösungsqualitäten bzw. Distanzen der besten gefundenen Lösung der 5 Testläufe von Instanz c5, g12, RC1_1000 in Abhängigkeit der Anzahl der Iterationen dargestellt. Hierbei werden die Auswirkungen des Abbruchkriteriums deutlich. Man kann erkennen, dass die GPU-Version in allen Fällen mehr Iterationen in der vorgegebenen Zeit ausführen kann.[118] Weiterhin wird ersichtlich, dass bei gleicher Iterationszahl die Lösungsgüte der GPU in etwa der der CPU-Version entspricht.[119]

Zur Beurteilung der beiden Verfahren aus diesen Grafiken müssten zwei Kostengeraden verwendet werden. Allerdings soll an dieser Stelle darauf verzichtet werden und ein anderer Ansatz Anwendung finden. Dazu werden in den Abbildungen 4.28 - 4.30 die Distanzen in Abhängigkeit der Geldmittel - sprich die Kosten, die sich bei Berechnung auf dem Cluster von Amazon.com ergeben - abgebildet. Daraus lässt sich ableiten, welche Lösungsgüte man für welchen eingesetzten Betrag in Rechenzeit bekommt. Es kommt nun im Vergleich zur Betrachtung der Iterationszahl zu einer Verschiebung. Das bedeutet man bekommt für einen Euro Rechenzeit unterschiedliche Lösungsgüten. Das resultiert daraus, dass zum einen die Iterationen unterschiedlich teuer sind und zum anderen unterschiedlich schnell abgeschlossen werden. So kostet eine Iteration auf der GPU zum Bearbeiten der Instanz g12 ca. 0,007

[114] Hier kann das Argument zum Tragen kommen, dass die Tesla M2050 lediglich 14 SMs besitzt und somit zur Berechnung von 16 Lösungen, wie es mit der GTX 580 ausgeführt wird, zu einer längeren Laufzeit kommt. Allerdings kann man diesem entgegnen, dass der Cluster, den man mietet, mit zwei Tesla M2050 ausgestattet ist, sodass man diesen Aspekt vernachlässigen kann. Das bedeutet, man könnte theoretisch 14 Lösungen auf der einen Grafikkarte und zwei auf der anderen berechnen lassen.

[115] Die Instanz kostet pro Stunde 1,61 USD. Hier wird ein Umrechnungskurs von 1,3 Dollar/Euro angesetzt.

[116] Die Hyperthreading-Technologie sorgt dafür, dass das Betriebssystem 16 Kerne statt lediglich 8, die tatsächlich vorhanden sind, sieht.

[117] Um genau zu sein, sieht Wendt (1995, S. 192) nicht die Zeit, sondern die Anzahl der Transitionen bzw. Iterationen als Produktionsfaktor an, wobei dieser ohne Einschränkungen durch die Zeit ersetzt werden kann.

[118] Das gilt auch, wenn die GPU-Zeiten entsprechend auf die Tesla-Karten umgerechnet werden.

[119] Als Iteration wird dabei ein vollständiger paralleler Durchlauf der VNS x SimA auf der GPU bzw. CPU angesehen. Das heißt, eine Iteration entspricht 100 Iterationen der VNS x SimA, die durchgeführt werden.

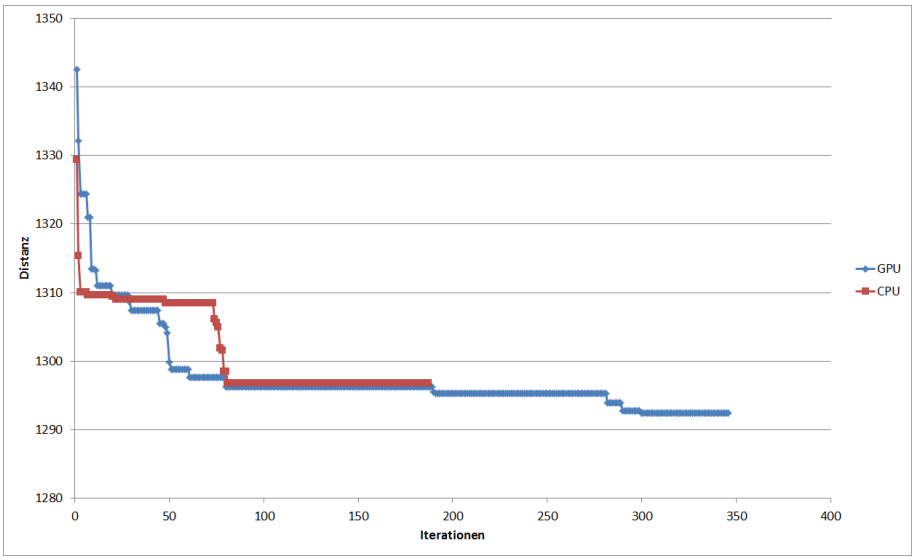


Abbildung 4.25: Distanz in Abhängigkeit der Iterationsanzahl (Instanz c5)

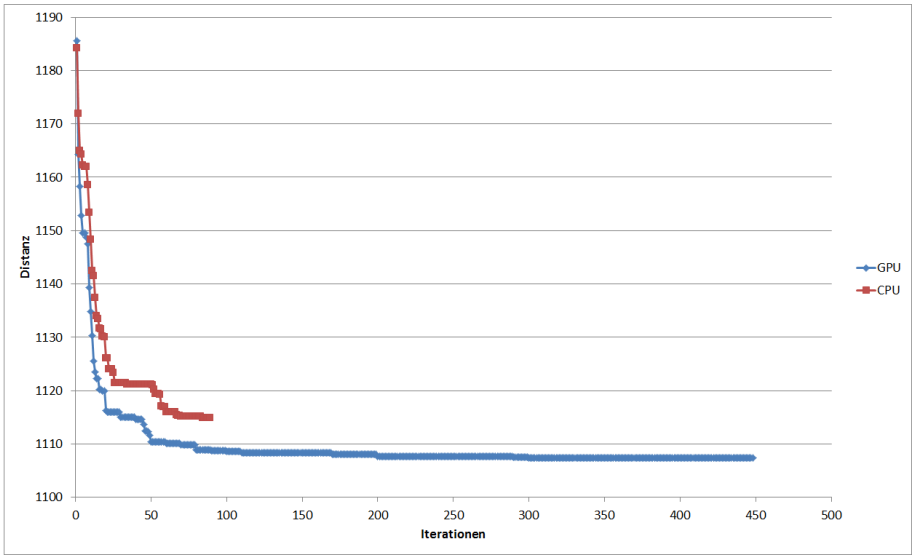


Abbildung 4.26: Distanz in Abhängigkeit der Iterationsanzahl (Instanz g12)

Euro wohingegen sie auf der CPU mit 0,014 Euro das Doppelte kostet.[120] Mit anderen Worten wird das Mehr, das die GPU-Version bei Amazon pro Stunde kostet, durch die schnellere Hardware und ihre effizientere Ausnutzung im Anwendungsfall dieser Arbeit kompensiert. In Tabelle 4.29 sind die Kosten pro Iteration nach den getesteten Instanzen aufgelistet. Man sieht, dass in allen Fällen die Verwendung einer GPU-Iteration weniger Kosten verursacht als die CPU-Iteration.

Es soll nun der Annahme von Wendt (1995, S. 193) gefolgt werden, dass die Distanz bzw. der resultierende Zielfunktionswert in Kilometer gemessen wird. Als Kilometerkosten werden lediglich die

[120] Hier gilt es noch zu beachten, dass die Rechenkapazitäten bei Amazon.com nur stundenweise und nicht auf Sekundenbasis gemietet werden können. Allerdings sollte durch die vereinfachende Annahme, dass die Rechenzeit auch sekundenweise gemietet werden kann, trotzdem ein Einblick in die wirtschaftliche Vorteilhaftigkeit eines Verfahrens gegeben sein.

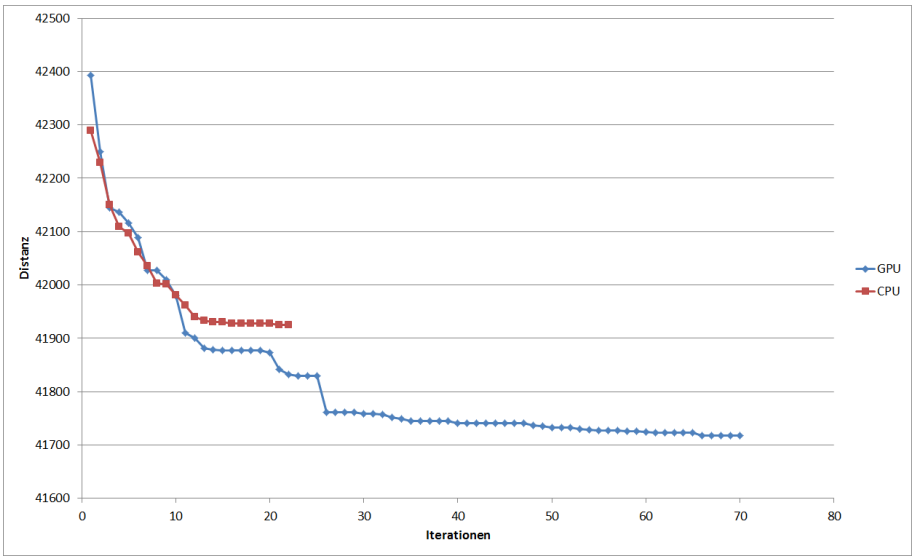


Abbildung 4.27: Distanz in Abhängigkeit der Iterationsanzahl (Instanz RC1_1000)

Instanz	CPU [Euro]	GPU [Euro]
c5	0,0022	0,0016
g12	0,0141	0,0070
RC1_1000	0,0567	0,0496

Tabelle 4.29: Kosten pro ausgeführter Iteration auf dem Cluster von Amazon.com

Kosten für Dieselkraftstoff herangezogen.[121] Es wird angenommen, dass ein Liter Diesel im Oktober 2012 für Großverbraucher 1,2 Euro kostet (vgl. Bundesverband Güterkraftverkehr Logistik und Entsorgung (BGL) e.V. (2012)). Legt man nun die Annahme zu Grunde, dass ein LKW auf 100 Kilometer ca. 40 Liter Kraftstoff verbraucht (Pander, 2012), ergibt sich ein Kilometerpreis von ca. 0,5 Euro. Das bedeutet, dass bei einem Einsatz von 0,5 Euro in Rechenzeit mindestens eine Kilometerreduzierung von 1 erzielt werden sollte, damit sich der Rechenaufwand lohnt.

Betrachtet man Abbildung 4.28 so wird deutlich, dass der Recheneinsatz bzw. Betrag von 0,4 Euro im Rahmen der CPU-Version für sich bereits gerechtfertigt ist, da eine erhebliche Kilometer- und somit Kostenreduzierung stattgefunden hat (über 30 Kilometer). Auch der Einsatz der GPU-Version lässt sich damit begründen. Wie es sich schon durch die Kosten pro Iteration in Tabelle 4.29 angekündigt hat, zeigt sich hier die Dominanz der GPU-Version. In dem gewählten Beispielfall findet die CPU-Version zunächst bessere Lösungen, wodurch sie zu bevorzugen wäre. Allerdings kehrt sich dieser Sachverhalt mit zunehmenden Einsatz von Rechenzeit bzw. Kosten ins Gegenteil um, zugunsten der GPU. Das bedeutet, wenn man bspw. bereit ist, 0,15 Euro in Rechenzeit zu investieren, resultiert daraus bei Verwendung der GPU eine Kilometerreduzierung von über 10 im Vergleich zur CPU. Auch bei maximalem Einsatz von 0,4 Euro in Rechenzeit kann die GPU-Version eine Kilometerreduzierung von ca. 1 gegenüber der

[121] Die realen Kosten, die u.a. Personalkosten, Mautgebühren oder Kosten für weitere Betriebsmittel wie z.B. Schmierstoffe enthalten, dürften ein Vielfaches darüber liegen.

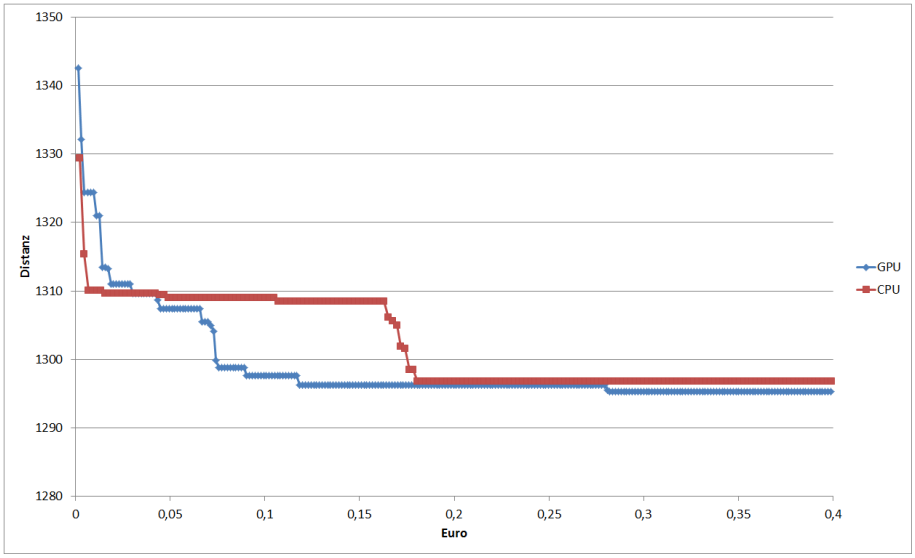


Abbildung 4.28: Distanz in Abhängigkeit der Geldmittel (Instanz c5)

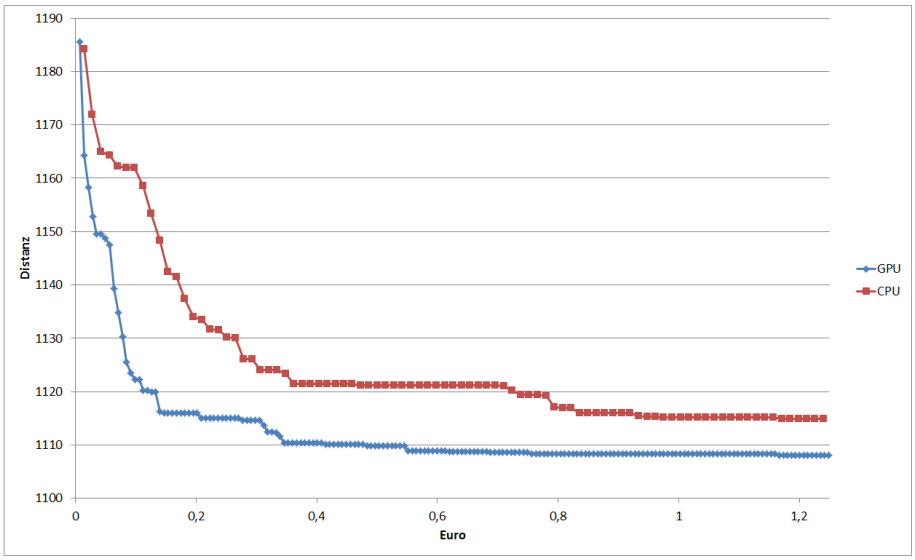


Abbildung 4.29: Distanz in Abhängigkeit der Geldmittel (Instanz g12)

CPU erzielen, was ihren Einsatz rechtfertigt.[122] Das gleiche Bild zeigt sich bei Betrachtung von Instanz g12. Hier ist in jedem Falle die GPU- der CPU-Version vorzuziehen. Bei maximaler Rechenzeit beträgt die Kilometerreduzierung der GPU-Version ca. 6. Im Fall von Instanz RC1_1000 kann man erst ab ca. 0,5 Euro, die für die Rechenzeit eingesetzt werden, von einem eindeutigen Vorteil auf Seiten der GPU sprechen. Allerdings führt die Grafikkarte bei maximalem Einsatz zu einer Reduzierung von über 150 Kilometern und somit zu einer theoretischen Kostenreduzierung von 75 Euro, was im Vergleich zum eingesetzten Kapital in die Rechenzeit von ca. 1,24 Euro eine sehr große Reduzierung darstellt.

In den Abbildungen 4.31, 4.32 und 4.33 finden sich die Regressionskurven, die aus den 5 Testdurch-

[122] In den Abbildungen 4.28 - 4.30 sind nicht alle Daten der GPU-Version dargestellt, da diese Version noch länger gelaufen ist als die CPU-Version. Es wird auf die Dauer der CPU-Version reduziert. Das resultiert daher, dass die GPU-Versionen auf die Rechenkapazitäten auf dem Cluster gemappt werden und somit die Maximaldauer $3600 \cdot 1,53$ Sekunden beträgt bzw. im Falle der Instanz c5 entsprechend weniger.

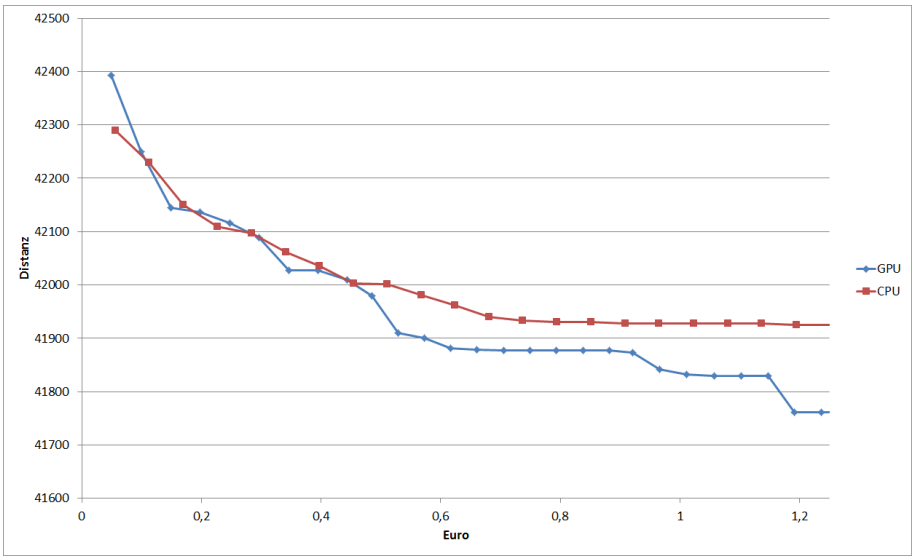


Abbildung 4.30: Distanz in Abhängigkeit der Geldmittel (Instanz RC1_1000)

läufen bei Betrachtung der jeweils besten Lösung zum entsprechenden Zeitpunkt bzw. Geldeinsatz resultieren. Auch hier zeigt sich der gleiche Verlauf wie zuvor gezeigt. Das bedeutet, bei einer geringen Rechendauer bzw. einem geringen Preis spielt es i.d.R. keine Rolle, ob die GPU verwendet wird oder nicht. Das kehrt sich mit zunehmendem Einsatz um. Je höher der monetäre Einsatz, desto vorteilhafter ist es, die GPU mit einzusetzen, da für den gleichen Geldeinsatz mehr Iterationen und damit eine bessere Lösungsgüte bzw. weniger Kilometer erreicht werden. Weiterhin spricht für eine längere, d.h. kostspieligere Verwendung der Cluster, dass damit die zu fahrenden Strecken erheblich reduziert und damit direkt Kosten eingespart werden können.

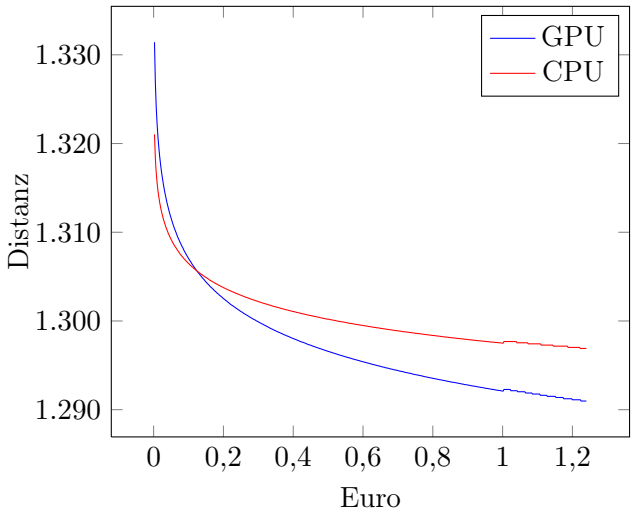


Abbildung 4.31: Regressionskurve der Distanz in Abhängigkeit der Geldmittel (Instanz c5)

Zusammenfassend kann man schlussfolgern, dass sich die Verwendung einer GPU und eine genügend lange Rechendauer positiv auf die zurückzulegenden Kilometer auswirken. Im betrachteten Beispiel der $VNS \times SimA_{GPU}$ und deren CPU-Pendant zeigt sich eindeutig, dass die GPU-Version präferiert werden sollte. Allerdings muss darauf hingewiesen werden, dass bei der GPU-Variante keine Entwicklungskosten berücksichtigt werden, die dazu führen könnten, dass sich das Bild zu Gunsten der CPU ändert.

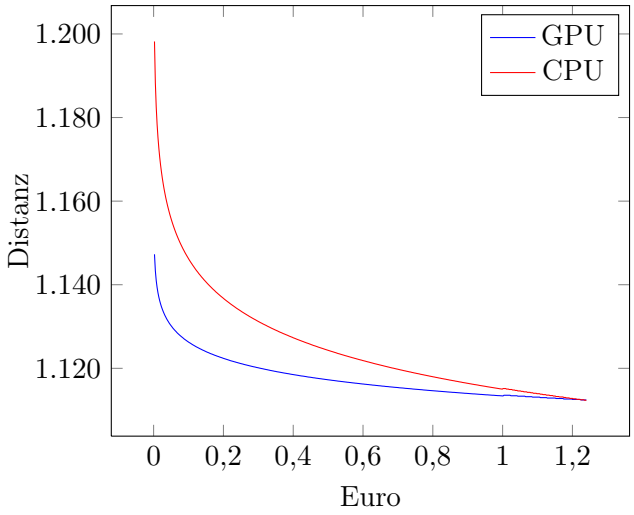


Abbildung 4.32: Regressionskurve der Distanz in Abhängigkeit der Geldmittel (Instanz g12)

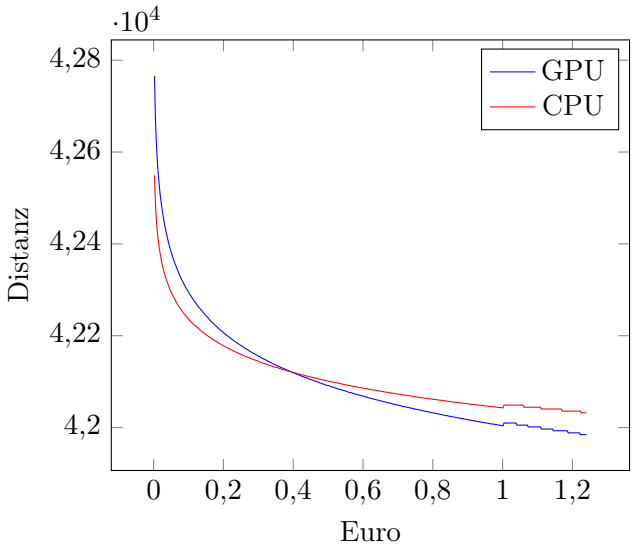


Abbildung 4.33: Regressionskurve der Distanz in Abhängigkeit der Geldmittel (Instanz RC1_1000)

Weiterhin dürfte jedoch der Kilometerpreis ein Vielfaches von dem betragen, was in dieser Arbeit angesetzt wird, so dass damit die Entwicklungskosten abschwächt werden.

5 Zusammenfassung, Ergebnisse und Ausblick

Zunächst soll eine Zusammenfassung über die Arbeit gegeben werden. Danach werden kurz die Ergebnisse skizziert, bevor sie mit einem Ausblick schließt.

5.1 Zusammenfassung

Hauptuntersuchungsgegenstand dieser Arbeit bilden parallele Algorithmen zur Lösung des Capacitated-Vehicle-Routing-Problem (CVRP). Zuerst werden die Grundlagen zu Tourenplanungsproblemen erläutert. Danach werden Lösungsverfahren, die zur Lösung des CVRPs - aber auch anderer Problemklassen - zum Einsatz kommen, beschrieben. Das Hauptaugenmerk liegt auf Verbesserungsverfahren und Metaheuristiken. Im Rahmen der Metaheuristiken werden das Simulated Annealing (SimA), die Tabusuche (TS), die Variable Neighborhood Search (VNS) und Genetische Algorithmen (GA) näher betrachtet, da sie in den Implementierungen, die dieser Arbeit zugrunde liegen, angewendet werden. Danach werden Charakteristika von parallelen Algorithmen vorgestellt. Neben allgemeinen Eigenschaften, die auf alle parallelen Anwendungen zutreffen, werden auch spezifische Charakteristika, die sich aus dem Kontext der Metaheuristiken heraus ergeben, betrachtet. Kennzahlen zum Vergleich paralleler und sequentieller Algorithmen werden ebenso vorgestellt. Nach der Erläuterung der Parallelisierung und der Charakterisierung nebenläufiger Anwendungen wird auf die Lösungsgüte und die Laufzeit, die ein Verfahren zum Berechnen von Resultaten benötigt, eingegangen. In diesem Kontext wird ebenfalls die Bedeutung der Benchmarks zum Vergleich von Metaheuristiken herausgestellt. Weiterhin findet sich in den Grundlagen ein Literaturüberblick für Vehicle-Routing-Probleme, wobei der Fokus auf parallelen Ansätzen bzw. GAs liegt. Letztendlich finden sich die Grundlagen der Hardwarearchitektur, die dieser Arbeit zugrunde liegt. Dazu wird zunächst die Hardwarearchitektur der Graphics Processing Units (GPU) von Nvidia beleuchtet, bevor das Programmiermodell Compute Unified Device Architecture (CUDA), das von Nvidia bereitgestellt wird und mittels dessen die GPUs des Herstellers angesprochen werden können, vorgestellt wird.

Nach Darstellung der Grundlagen finden sich konkrete Beschreibungen der implementierten Software in dieser Arbeit. Dazu werden zunächst die lokalen Suchoperatoren und deren Umsetzung erläutert. Verwendet werden der Relocate-, Swap-, Or-Opt-, 2-Opt*-, 2-Opt- und Cross-Exchange-Operator. Neben den Datenstrukturen wird auch auf die Spezifika einer Tabuliste im Rahmen der Operatoren eingegangen. Weiterhin erfolgen die Beschreibungen aus Sichtweise der GPU und somit der Parallelisierung.

Gleiches gilt für die Beschreibungen der Metaheuristiken, die von den lokalen Suchoperatoren Gebrauch machen. Hierbei wird zunächst eine VNS in Kombination mit einem SimA umgesetzt. Neben einer Version, die viele Aufgaben auf der CPU und nur wenige auf der GPU löst (CPU-Version), wird auch eine GPU-Version implementiert, die theoretisch vollkommen autark ohne Eingriffe der CPU, CVRPs lösen kann. Bei der GPU-Version handelt es sich um ein Multistart-Verfahren, bei dem mehrere

Lösungen parallel mit einer VNS und SimA bearbeitet werden. In Analogie dazu entsteht eine TS, die ebenfalls Gebrauch von den lokalen Suchoperatoren macht. Auch hier wird eine CPU- und eine GPU-Version umgesetzt. Im Rahmen der VNS mit SimA werden weiterhin noch zwei Elemente zur Variation entwickelt. Zum einen handelt es sich um einen adaptiven Abkühlungsplan, der vollständig von der GPU verwaltet wird, und zum anderen um eine Shaking-Prozedur, die dem Cooperative Simulated Annealing (COSA) entlehnt ist. Schließlich finden die TS sowie die VNS mit SimA Anwendung im Kontext eines GA. Dazu werden sie an der Stelle des Mutationsoperators im GA eingesetzt, weshalb man in diesem Fall auch von einem memetischen Algorithmus (MA) sprechen kann. Mittels der in den Grundlagen erläuterten Merkmalen für Metaheuristiken und paralleler Algorithmen, wird ein Vergleich der Verfahren mit der Literatur sowie untereinander vorgenommen.

Nachdem die Grundlagen und die Implementierung vorgestellt sind, wird abschließend mit numerischen Analysen die Wirkung der Algorithmen untersucht. Zunächst liegt der Fokus auf den lokalen Suchoperatoren und der Frage, welche Speichertypen der Grafikkarte zur Ablage der Distanzmatrix genutzt werden sollten. Weiterhin werden die Auswirkungen des Einsatzes einer Tabuliste auf die Laufzeiten untersucht. An dieser Stelle findet auch ein Vergleich mit einer reinen CPU-Implementierung statt, d.h., es werden Kennzahlen wie der Speedup ermittelt. Danach werden die Metaheuristiken auf unterschiedlichen Benchmarkinstanzen getestet. Die Benchmarksätze stammen von Golden et al. (1998), Christofides et al. (1979), Taillard (1993) und Gehring und Homberger (1999). Mittels der so ermittelten Resultate findet eine Positionierung der implementierten Verfahren untereinander sowie mit Verfahren der Literatur statt. Abschließend wird die Wirtschaftlichkeit einer reinen GPU-basierten Metaheuristik mit ihrem Pendant, das nur auf der CPU ausgeführt wird, analysiert. Dazu wird als Referenz für die Kosten von Rechenzeit auf die Preise, die sich beim Mieten von Rechnerkapazitäten auf dem Cluster von Amazon.com ergeben, abgestellt. Weiterhin wird die CPU-Version auf diesem Cluster gestartet.

5.2 Ergebnisse

Die Ziele der Arbeit gehen in verschiedene Richtungen. So sollen durch die Analyse von Operatoren und ihrer Ausführungen Gestaltungsempfehlungen gegeben werden können, wie eine möglichst performante Implementierung aussehen kann. Weiterhin soll die einfache Nutzung der lokalen Suchoperatoren im Kontext von Metaheuristiken zeigen, ob man mit State-of-the-Art-Algorithmen konkurrieren kann. Gleichzeitig soll hierbei die unterschiedlich starke Nutzung von der GPU und deren Auswirkung auf die Lösungsgüte berücksichtigt werden. Letztendlich gilt es herauszufinden inwiefern der Einsatz von GPUs im Vergleich zur reinen CPU-Nutzung Vorteile hinsichtlich der Wirtschaftlichkeit erbringen kann. Außerdem ist dem Autor dieser Arbeit kein Beitrag, der die Grafikkarte in einem Algorithmus zum Lösen des CVRPs einsetzt, bekannt und der konkrete Ergebnisse zu bekannten Benchmarkinstanzen des CVRPs der Literatur liefert. Mit dieser Arbeit ist es möglich, erste Vergleiche von GPU-basierten Verfahren mit bekannten Verfahren der Literatur anzustellen.

Bei der Analyse der lokalen Suchoperatoren wird deutlich, dass man eine Reduktion der Laufzeiten erreichen kann, wenn man die Distanzmatrix einmalig zum Programmstart erstellt und danach im Texturspeicher der Grafikkarte ablegt, um im Folgenden darauf zuzugreifen. Es zeigt sich, dass keine allgemeingültige Aussage bzgl. der Anzahl der Threads, die pro Block gestartet werden sollten, möglich ist. Selbst bei Betrachtung eines einzelnen Operators hängt die ideale Threadanzahl von der Knotenanzahl

der Problemstellung ab. Damit ist zu empfehlen, dass die Threadzahl pro Block sich zu Beginn einer Metaheuristik selbst einstellt. Außerdem zeigt sich, dass die Tabuliste sowie deren Länge Einfluss auf die Kernelzeit bei Bewertung eines Moves nimmt.

Schließlich werden im Rahmen der Betrachtung der lokalen Suchoperatoren Vergleiche mit einer Referenz-CPU-Implementierung vorgenommen. Hier zeigt sich, dass bei beiden Implementierungen und unabhängig vom Operator die Ausführungsdauer mit der Anzahl der Knoten im CVRP wächst. Allerdings steigt die Ausführungszeit im Rahmen der GPU langsamer an als die der CPU, sodass insbesondere bei Betrachtung großer Instanzen empfohlen wird, die GPU zu verwenden. Es können in dieser Arbeit Speedups im Vergleich zur CPU-Umsetzung bis zu ca. 16 erzielt werden. Bei Betrachtung weiterer Kennzahlen wie der Effizienz oder Kostenoptimalität wird deutlich, dass diese nicht ohne weitreichende Annahmen bei Verwendung einer Grafikkarte bei der Parallelisierung genutzt werden können. Deshalb drängt sich der Speedup als Referenz förmlich auf, da damit keine Annahmen bzgl. der Prozessoranzahl getroffen werden müssen.

Im Rahmen der Metaheuristiken zeigt sich, dass mit vergleichsweise einfachen Mitteln sehr gute Resultate erzielbar sind. Die beiden Verfahren $VNS \times SimA_{CPU}$ und TS_{CPU}, die einen Großteil der Berechnungen auf der CPU ausführen, können bereits mit sehr guten Ergebnissen über alle Benchmarkinstanzen hinweg aufwarten. Hierbei werden mehrere neue beste Lösungen für die Instanzen von Gehring und Homberger (1999) ermittelt. Durch extensive Nutzung der GPU ist es durch Multistart-Verfahren möglich, die Lösungsgüte weiter zu steigern. Des Weiteren zeigt sich in diesem Kontext, dass die Verwendung eines adaptiven Abkühlungsplans keine bzw. nur sehr geringe Auswirkungen auf die Ausführungsdauer eines Kernels besitzt, weshalb grundsätzlich darüber diskutiert werden sollte, einen solchen zu verwenden, um mit dem Algorithmus flexibler auf unterschiedlichste Problemstellungen reagieren zu können. Auch die Verwendung von einer an COSA-angelehnten Methode zum Shaking wirkt sich nur marginal auf die Kernelzeit aus.

Unter Hinzunahme der Lösungsverfahren, die auf einer Lokalsuche beruhen, in einen GA kann eine weitere Leistungssteigerung hinsichtlich der Lösungsgüte erreicht werden. Hier werden die höchsten Lösungsqualitäten erreicht. Die GAs, die sich eine Variante der VNS mit SimA zu Nutze machen, dominieren im Kontext der Instanzen von Golden et al. (1998) alle anderen Verfahren. Auch bei Betrachtung der Gehring-Benchmarks erzielen sie die beste Lösungsgüte, wobei der GA, der die $VNS \times SimA_{GPU}$ mit COSA im Shaking verwendet, am besten abschneidet und insgesamt 13 neue beste Lösungen ermitteln kann. Vergleicht man die Verfahren mit welchen der Literatur, wird deutlich, dass sie bereits sehr gute Ergebnisse bei Betrachtung der Golden-Instanzen nach sehr kurzer Rechenzeit finden und viele andere Verfahren dominieren. Bezüglich der Lösungsgüte sind nur wenige Algorithmen der Literatur besser. Die besseren verwenden oftmals komplexe Verfahren, um diese Lösungsqualität zu erreichen. Demgegenüber stehen die Implementierungen dieser Arbeit, die sich aus einfachen Operatoren zusammensetzen und dennoch mit den besten Verfahren konkurrieren können.

Um Rechnerkapazitäten im Cluster von Amazon.com zu mieten, fällt zunächst der höhere Preis auf, den man für Rechner mit GPU-Hardware zahlen muss. Dieser relativiert sich jedoch bei Betrachtung der Iterationszahl, die mittels einer GPU-Version und einer reinen CPU-Version erzielt wird. So kostet eine Iteration auf einem GPU-Rechner teilweise halb so viel wie eine Iteration auf dem CPU-Rechner. Durch den geringen Preis pro Iteration ist es möglich, mehr Iterationen auf der GPU durchzuführen und somit signifikant bessere Lösungsgüten zu erhalten.

Zusammenfassend lässt sich schlussfolgern, dass es von großer Bedeutung ist, die Programme auf der GPU entsprechend der zugrunde liegenden Hardware zu optimieren. Nur so ist eine optimale Ausnutzung der vorhandenen Hardwareressourcen möglich. Weiterhin zeigt sich, dass durch die Verwendung der GPU und damit dem Mehr an Rechenkapazität vergleichsweise einfache Verfahren mit komplexer zu implementierenden Verfahren der Literatur konkurrieren können. So war es möglich, insgesamt 16 bzw. 21 neue beste Lösungen für die Gehring-Instanzen zu ermitteln.[123]

5.3 Ausblick

Der Bereich der lokalen Suchoperatoren auf der GPU für CVRP wird weitestgehend durch den Beitrag von Schulz (2013) und dieser Arbeit abgedeckt. In weiteren Forschungen sollte untersucht werden, wie diese Operatoren in anderen Heuristiken bzw. Metaheuristiken eingebunden werden können und welche Ergebnisse damit erzielbar sind. Der Grundstein hierzu wird in dieser Arbeit mit vergleichsweise einfachen Verfahren gelegt. Nun sollte evaluiert werden, inwiefern sich die Lösungsgüte ändert, wenn komplexer zu implementierende Algorithmen der Literatur die lokalen Suchoperatoren bzw. ihre Umsetzung auf der GPU verwenden.

Ein weiterer Aspekt, der untersucht werden könnte, ist die Liste bzw. die Inhalte der erzeugenden Kanten. Die lokalen Suchoperatoren dieser Arbeit entstehen aus Generatorkanten. Damit ist es möglich, bereits zu Beginn der Berechnungen, viele Kanten aus dieser Menge zu entfernen und somit den Aufwand bei Durchsuchen der Nachbarschaft zu reduzieren. Die Idee dazu stammt von Toth und Vigo (2003). Es sollte betrachtet werden, inwiefern sich die Lösungsgüte durch Reduzierung der Anzahl an Generatorkanten in den in der Arbeit umgesetzten Metaheuristiken verändert. Einhergehend mit der Reduzierung der Kantenanzahl und der damit zu erwartenden Reduzierung der Laufzeit, könnten größere Instanzen betrachtet werden.

Weiterhin sollten Untersuchungen bzgl. der Parameter vorgenommen werden. Hierbei gilt es insbesondere die Größe der Population zu evaluieren, da in der Arbeit mit einer vergleichsweise kleinen Population gearbeitet wurde. Die Auswirkungen auf die Laufzeit wurden bereits im Groben skizziert. Das heißt, auch hier sollten die Auswirkungen auf die Lösungsgüte im Vordergrund stehen.

Auch die Nutzung neuerer Hardware bzw. des neuen Frameworks CUDA 5.0 sollte in zukünftigen Untersuchungen Berücksichtigung finden. Eine der größten Neuerungen im CUDA-Framework ist die Unterstützung des sog. „Dynamic Parallelism“, der es ermöglicht aus einem Thread auf der GPU heraus einen neuen Kernel zu starten. Die entsprechende Hardware dazu erscheint voraussichtlich im Frühjahr 2013 (Fischer, 2012). Hierbei könnte untersucht werden, wie sich eine weitere Entkoppelung der GPU von CPU der hier vorgestellten Algorithmen auf die Laufzeit auswirkt. Eventuell sind somit nach einmaligem Kernelstart von der CPU aus überhaupt keine weiteren Eingriffe von selbiger notwendig.

Wie beschrieben liegt die Performance von CUDA-Anwendungen bei direktem Vergleich des OpenCL-Pendants auf Nvidia-GPUs vorne. Dennoch könnte eine Portierung des bestehenden CUDA-Codes zu OpenCL interessante Einsichten ermöglichen. Auch wenn es bereits Untersuchungen zwischen CUDA- und OpenCL-Performance gibt, könnten in diesem Fall die Auswirkungen für den konkreten

[123] Bei der Analyse der Ergebnisse von Mester und Bräysy (2007), die die bisher einzigen Ergebnisse zu den Gehring-Instanzen veröffentlicht haben, wurde herausgefunden, dass die Lösungen, die zu den Instanzen der Klasse C2 angegeben sind, nicht gültig sind, da zu wenige Fahrzeuge verwendet werden.

Fall der Tourenplanung betrachtet werden. Weiterhin möglich ist dann der Einsatz auf alternativen Hardwareplattformen wie die von AMD/ATI. Hier verspricht die theoretisch höhere Leistungsfähigkeit der Grafikkarten von AMD/ATI weitere Laufzeitverbesserungen der hier vorgestellten Algorithmen.

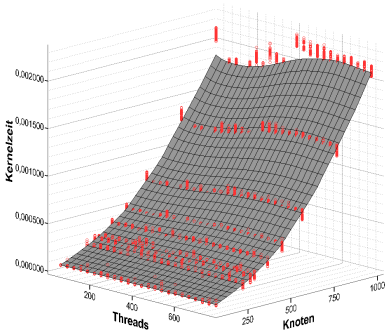
Neben den Grafikkarten-Herstellern sollte auch Intel nicht außer Acht gelassen werden. Durch die Veröffentlichung des Intel Xeon Phi Coprozessors versucht Intel in das Marktsegment der Beschleunigerkarten, das bisher weitestgehend von Nvidia durch die Grafikkarten beherrscht wird, einzutreten (Stiller, 2012). Auch hierbei sollte das Hauptaugenmerk auf Betrachtung der Veränderung der Ausführungsdauer liegen.

Gerade die soeben aufgeführten neuen Hardwareplattformen weisen eine große Diversifität auf. Die Vergleiche zwischen den Plattformen über bekannte Kennzahlen sind - vom Speedup einmal abgesehen - nicht ohne Weiteres möglich, weshalb hier ein Entwicklungsbedarf an neuen Kennzahlen besteht. Weiterhin ist die Beurteilung der Wirtschaftlichkeit mit steigender Zahl an Plattformen zunehmend schwerer zu realisieren. In dieser Arbeit war es möglich, auf den Cluster eines Drittanbieters abzustellen und somit die Kosten für Rechenzeit einschätzen zu können. Das ist bei AMD/ATI-GPUs bzw. den Koprozessoren von Intel nicht ohne Weiteres möglich, weshalb an dieser Stelle neue Kennzahlen entwickelt werden könnten, um die Wirtschaftlichkeit - auch unabhängig von einem Drittanbieter - zu bestimmen. Hierbei gilt es unter anderem zu evaluieren, welche Faktoren berücksichtigt werden müssten, um die Kosten für Rechenzeit zu ermitteln.

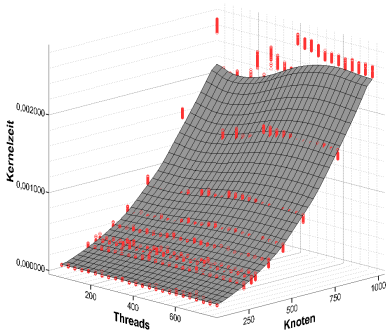
A Abbildungen

A.1 Kernelzeiten der Operatoren in Abhängigkeit der Thread- und Knotenanzahl

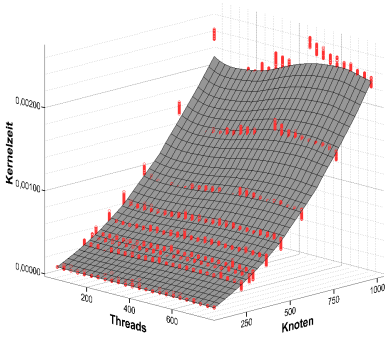
In diesem Abschnitt sind die Kernelzeiten, die sich in Abhängigkeit der Thread- und Knotenzahl ergeben, dargestellt (vgl. hierzu Kapitel 4.1.1.1).



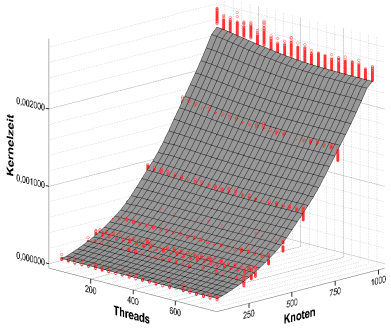
(a) Koordinaten in Globalspeicher



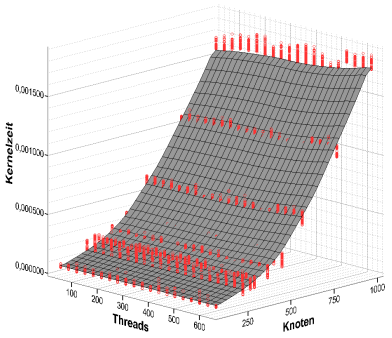
(b) Koordinaten in Constant Memory



(c) Koordinaten in Texturspeicher

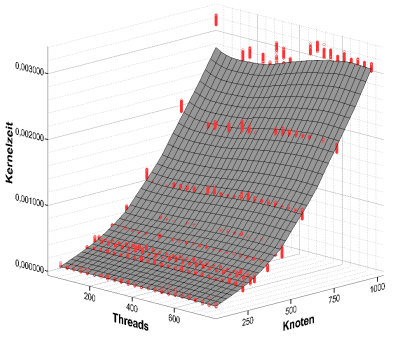


(d) Distanzmatrix in Globalspeicher

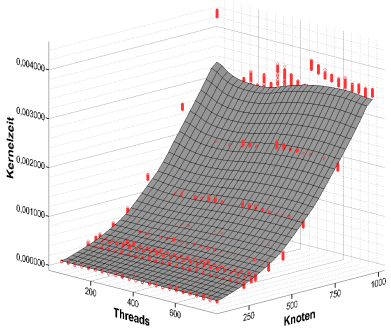


(e) Distanzmatrix in Texturspeicher

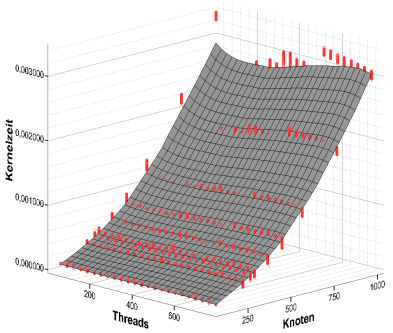
Abbildung A.1: Kernelzeiten des Swap-Operators



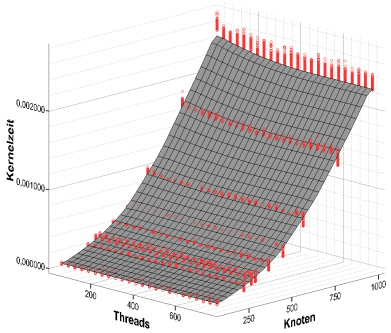
(a) Koordinaten in Globalspeicher



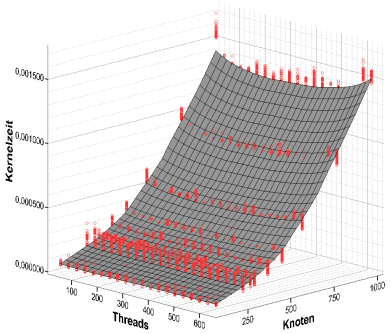
(b) Koordinaten in Constant Memory



(c) Koordinaten in Texturspeicher

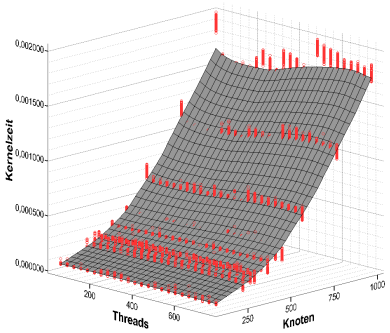


(d) Distanzmatrix in Globalspeicher

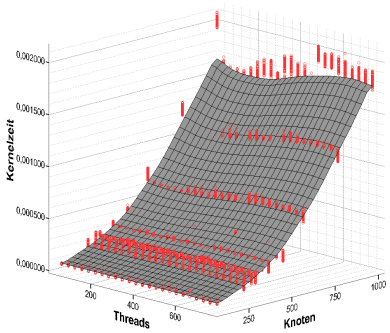


(e) Distanzmatrix in Texturspeicher

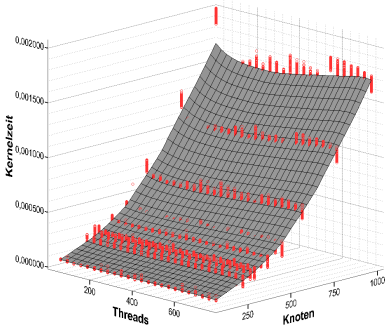
Abbildung A.2: Kernelzeiten des Or-Opt-Operators



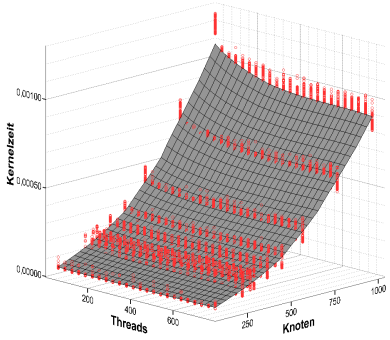
(a) Koordinaten in Globalspeicher



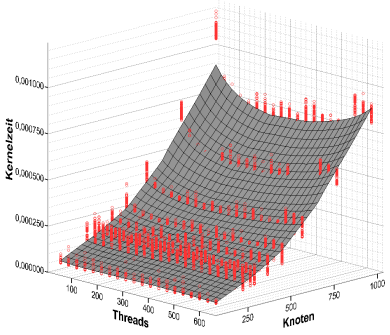
(b) Koordinaten in Constant Memory



(c) Koordinaten in Texturspeicher

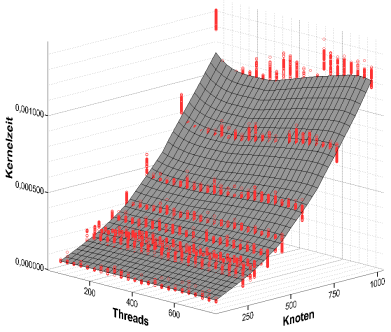


(d) Distanzmatrix in Globalspeicher

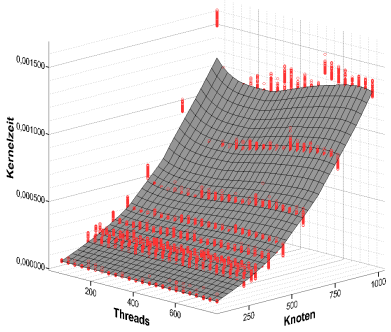


(e) Distanzmatrix in Texturspeicher

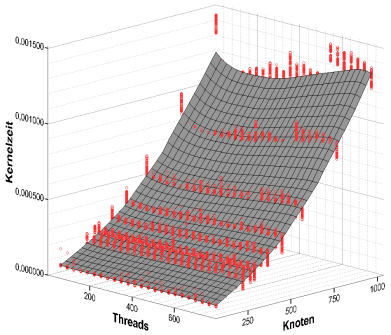
Abbildung A.3: Kernelzeiten des 2-Opt*-Operators



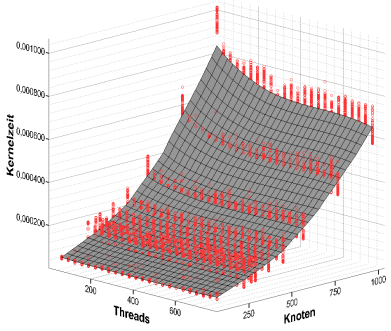
(a) Koordinaten in Globalspeicher



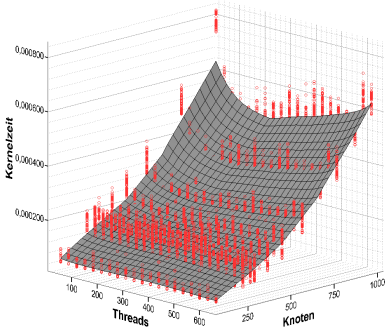
(b) Koordinaten in Constant Memory



(c) Koordinaten in Texturspeicher



(d) Distanzmatrix in Globalspeicher



(e) Distanzmatrix in Texturspeicher

Abbildung A.4: Kernelzeiten des 2-Opt-Operators

A.2 Speedups der lokalen Suchoperatoren

Die Speedups, die sich durch Anwendung der lokalen Suchoperatoren und im Vergleich zu einer CPU ergeben, sind im Folgenden abgebildet (vgl. hierzu Abschnitt 4.1.1.3).

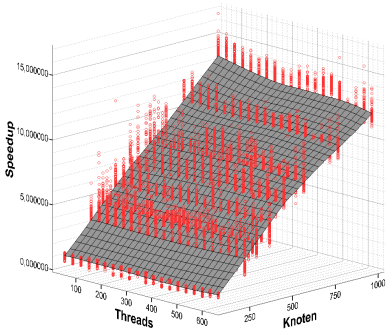


Abbildung A.5: Speedup bei Anwendung des Swap-Operators

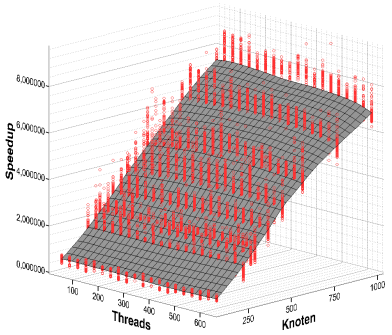


Abbildung A.6: Speedup bei Anwendung des Or-Opt-Operators

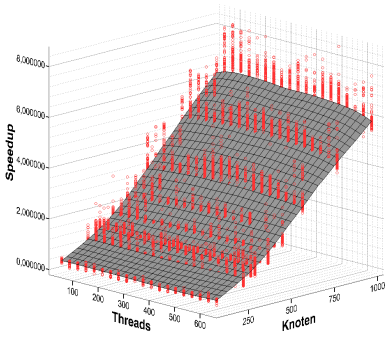


Abbildung A.7: Speedup bei Anwendung des 2-Opt*-Operators

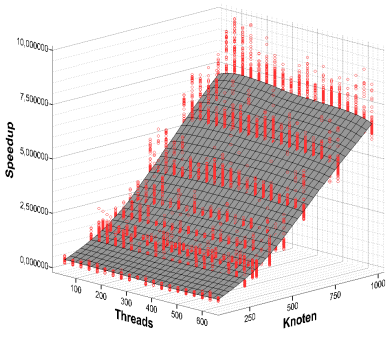
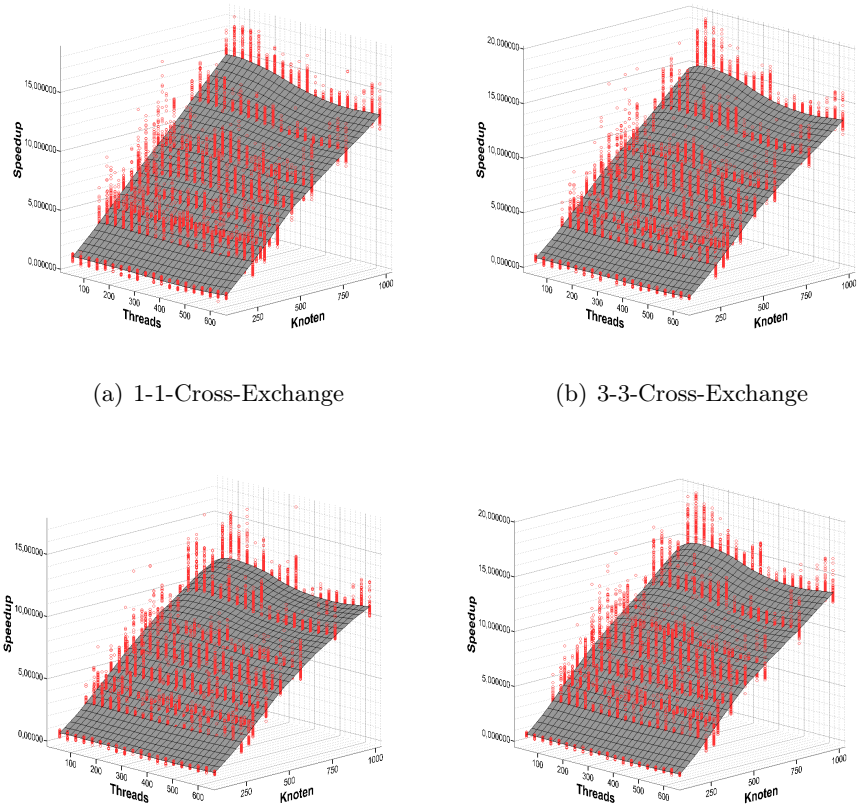


Abbildung A.8: Speedup bei Anwendung des 2-Opt-Operators



(a) 1-1-Cross-Exchange

(b) 3-3-Cross-Exchange

(c) 1-5-Cross-Exchange

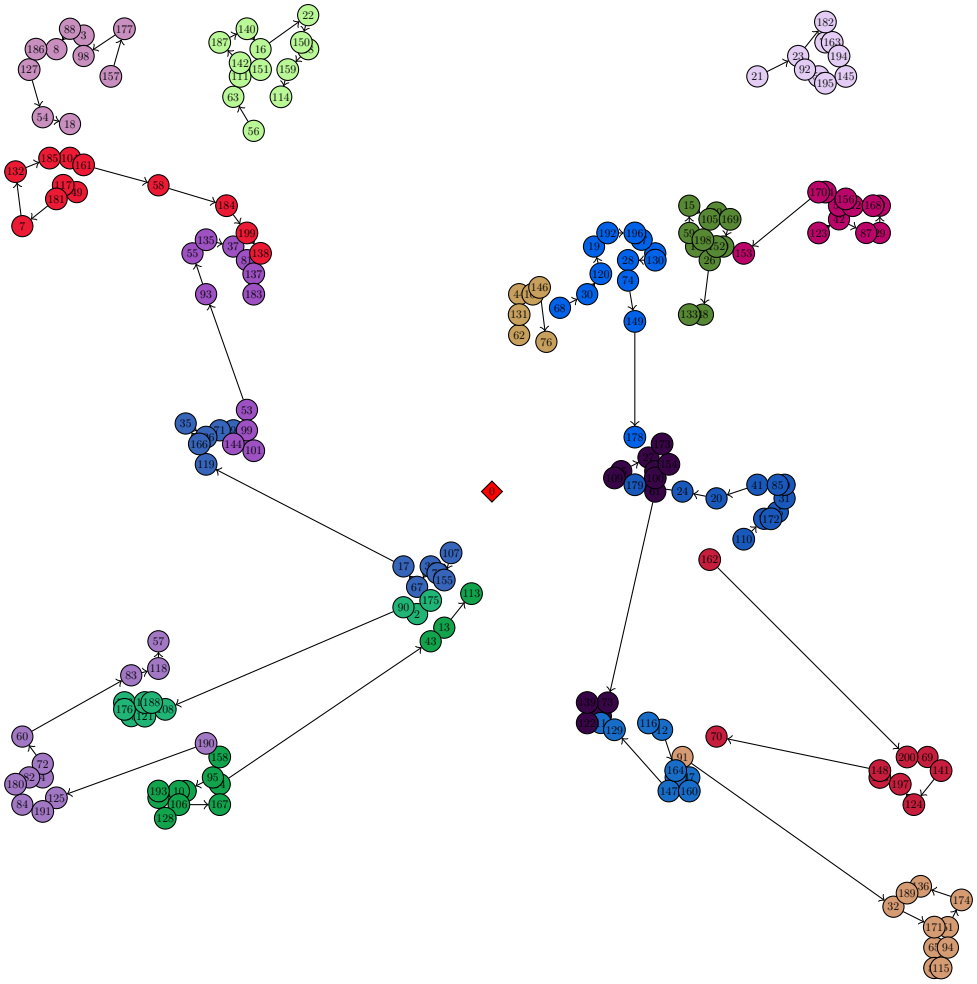
(d) 5-5-Cross-Exchange

Abbildung A.9: Speedup bei Anwendung des Cross-Exchange-Operators

A.3 Neue beste Lösungen der Gehring-Instanzen

In diesem Abschnitt finden sich die neuen besten Lösungen der Benchmarkinstanzen von Gehring und Homberger (1999), die im Rahmen dieser Arbeit gefunden wurden. Die grafische Darstellung soll lediglich als Anhaltspunkt dienen wie die grobe Struktur der Lösungen aussieht. Viele Knoten liegen unmittelbar übereinander, sodass dadurch nicht die Lösung rekonstruiert werden kann. Deshalb finden sich in unmittelbarer Folge auf die Abbildungen die Lösungen in Tabellenform. Das Depot ist als Raute dargestellt. Die unterschiedlichen Farben der Knoten repräsentieren die Zugehörigkeit zu einer Route, haben ansonsten jedoch keine Bedeutung. Auch lassen sich aus gleichfarbigen Routen zweier unterschiedlicher Instanzen keine Rückschlüsse auf die Lösung ziehen, da die Farben pro Instanz zufällig gewählt sind. Die Kanten vom Depot ausgehend bzw. zum Depot hinführend werden der Übersichtlichkeit halber nicht angezeigt.

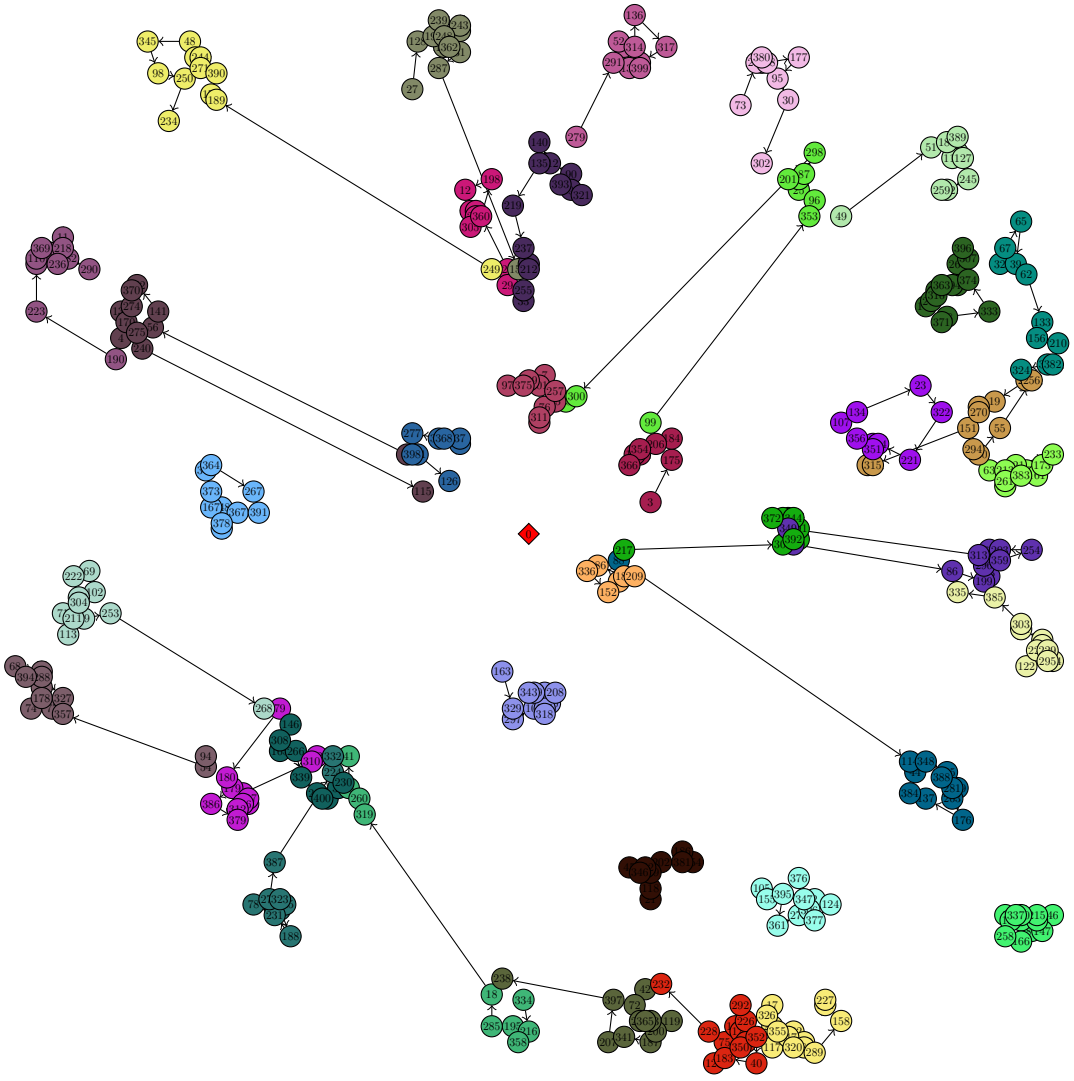
Instanz C1_200



Knotenanzahl	201
Maximalkapazität	200
Distanz	2567,67
Routenanzahl	18
Route 1	0 14 198 59 15 105 89 169 40 152 26 48 133 0
Route 2	0 21 23 182 75 163 194 145 195 52 92 0
Route 3	0 49 117 181 7 132 185 104 161 58 184 199 138 0
Route 4	0 56 63 111 151 142 187 140 16 22 150 38 159 114 0

Route 5	0 62 131 44 102 146 76 0
Route 6	0 68 30 120 19 192 196 97 96 130 28 74 149 178 0
Route 7	0 91 32 171 65 86 115 94 51 174 136 189 0
Route 8	0 101 144 1 99 53 93 55 135 37 81 137 183 0
Route 9	0 107 155 78 39 67 17 119 166 35 126 71 9 0
Route 10	0 109 45 27 173 154 64 100 61 73 6 122 139 0
Route 11	0 110 77 172 25 31 80 85 41 20 24 179 0
Route 12	0 116 12 164 66 47 160 147 129 11 0
Route 13	0 123 42 87 29 79 168 112 156 50 134 170 153 0
Route 14	0 157 177 98 3 88 8 186 127 54 18 0
Route 15	0 158 95 5 10 193 46 128 106 167 34 43 13 113 0
Route 16	0 162 200 69 141 124 197 103 148 70 0
Route 17	0 175 2 90 108 121 36 176 143 33 165 188 0
Route 18	0 190 125 191 84 180 82 4 72 60 83 118 57 0

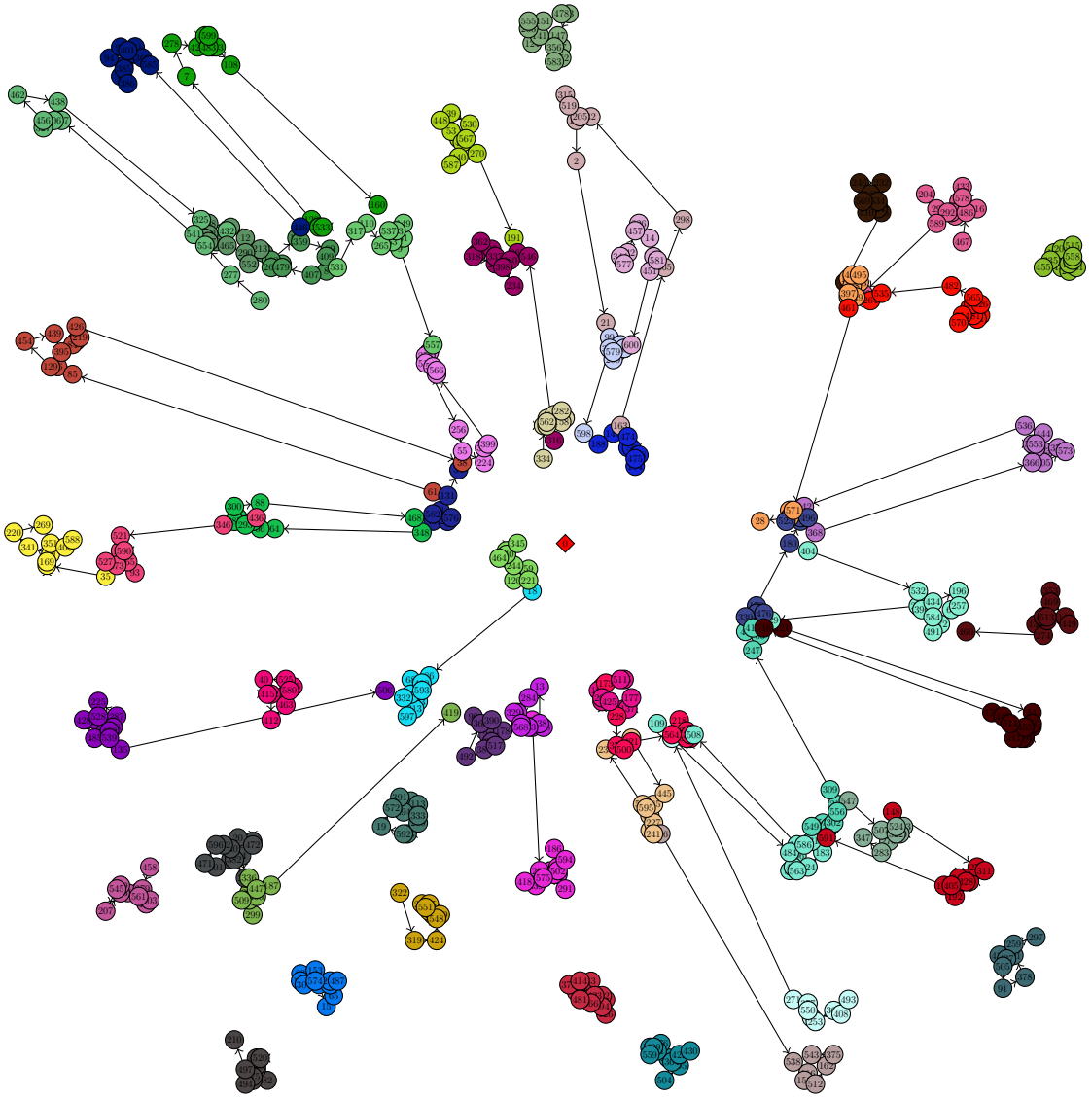
Instanz C1_400



Knotenanzahl	401
Maximalkapazität	200
Distanz	6721,10
Routenanzahl	37
Route 1	0 3 175 184 206 354 83 150 366 0
Route 2	0 5 160 337 330 143 153 215 46 147 89 166 258 0
Route 3	0 17 326 214 355 117 247 162 320 168 283 289 158 104 227 0
Route 4	0 20 37 368 325 277 398 331 126 0
Route 5	0 27 128 192 239 243 171 248 108 362 31 287 157 0
Route 6	0 29 28 360 198 12 9 47 299 305 0
Route 7	0 32 67 65 39 62 133 156 210 382 338 324 0

Route 8	0 42 72 273 365 203 119 200 187 341 207 397 238 0
Route 9	0 49 51 181 389 1 116 127 245 242 259 0
Route 10	0 63 213 121 233 173 161 50 383 261 131 0
Route 11	0 66 56 141 252 370 274 138 170 4 275 240 115 0
Route 12	0 73 265 85 380 328 177 95 30 302 0
Route 13	0 79 180 179 386 379 312 246 81 197 84 310 92 0
Route 14	0 88 114 44 348 235 388 281 129 263 176 137 384 0
Route 15	0 94 54 357 77 74 178 6 394 68 14 288 327 0
Route 16	0 97 375 149 101 7 257 15 76 311 241 0
Route 17	0 99 353 96 25 70 87 298 201 300 13 0
Route 18	0 102 69 222 159 304 71 113 211 59 253 268 0
Route 19	0 105 155 395 361 272 306 377 124 2 347 376 0
Route 20	0 107 134 23 322 221 284 351 58 356 0
Route 21	0 122 220 103 295 204 229 93 24 144 303 385 335 0
Route 22	0 146 308 165 266 339 262 400 8 34 230 0
Route 23	0 163 329 297 106 318 145 100 208 64 269 343 0
Route 24	0 186 154 381 202 342 309 118 21 346 45 0
Route 25	0 190 223 35 110 369 11 218 61 236 182 290 0
Route 26	0 217 301 139 392 36 191 344 91 130 372 0
Route 27	0 249 189 174 390 271 244 57 48 345 98 250 234 0
Route 28	0 279 291 52 16 314 136 317 22 399 132 0
Route 29	0 280 86 199 82 296 359 254 293 60 53 313 349 0
Route 30	0 292 226 109 172 26 350 282 352 40 183 123 75 228 232 0
Route 31	0 294 120 55 256 205 19 193 270 151 315 80 0
Route 32	0 321 196 393 90 112 140 135 219 237 169 212 255 33 0
Route 33	0 323 225 125 188 231 78 278 387 224 332 0
Route 34	0 334 216 358 195 285 18 319 260 43 41 0
Route 35	0 336 152 10 209 185 286 0
Route 36	0 371 38 333 374 307 396 264 111 194 363 251 316 276 142 0
Route 37	0 391 367 164 378 148 167 373 340 364 267 0

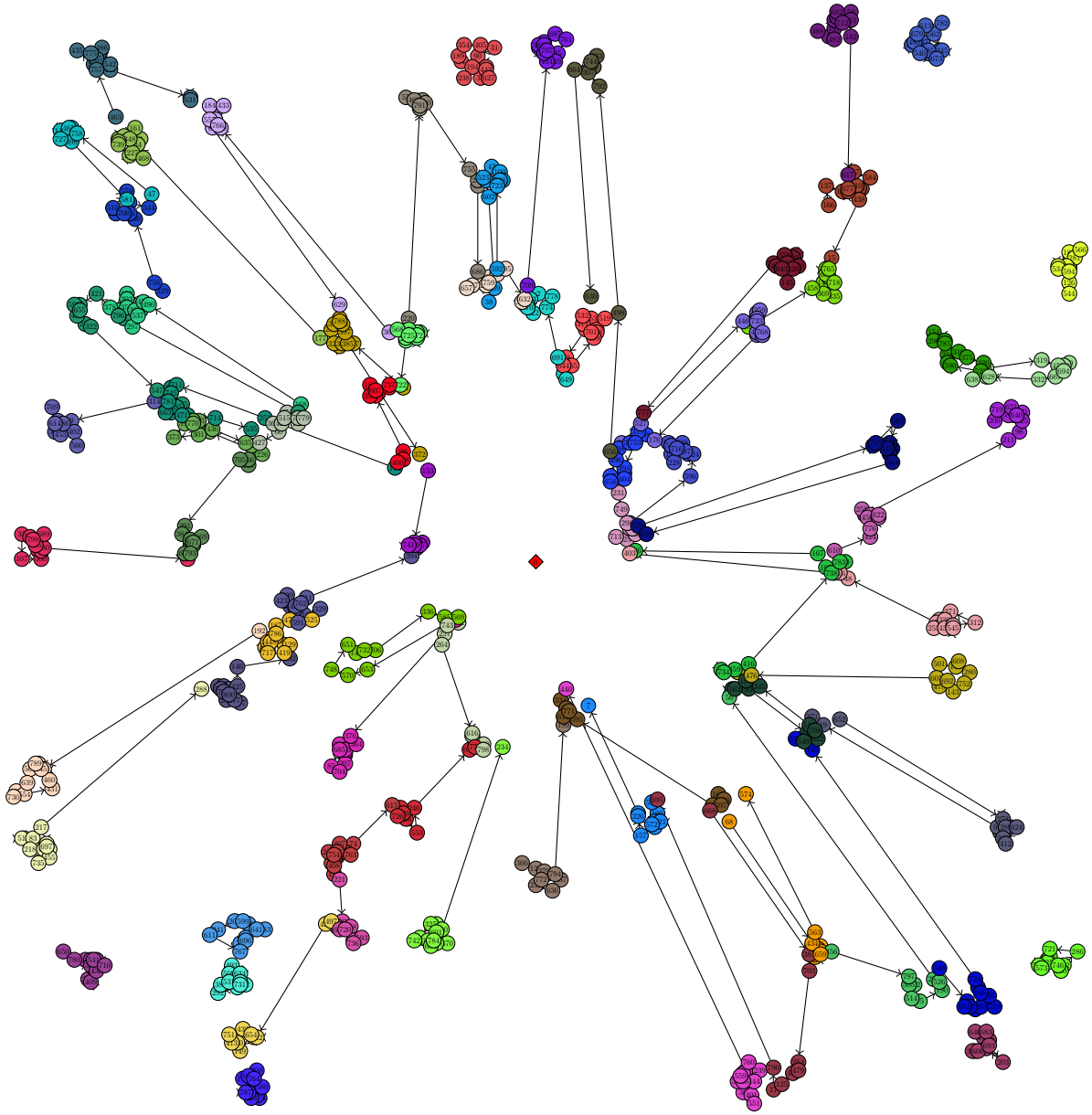
Instanz C1_600



Knotenanzahl	601
Maximalkapazität	200
Distanz	13436,00
Routenanzahl	56
Route 1	0 11 472 24 255 382 122 201 471 596 542 540 20 0
Route 2	0 18 86 490 593 223 313 597 332 331 30 68 0
Route 3	0 35 26 169 49 341 220 269 351 400 588 0
Route 4	0 43 423 102 80 126 394 466 272 481 72 87 370 414 0
Route 5	0 61 85 46 129 454 439 395 84 1 219 426 38 0
Route 6	0 76 208 418 239 575 92 214 291 100 502 389 594 186 0
Route 7	0 109 484 156 473 89 563 324 183 431 586 107 508 0
Route 8	0 113 333 260 193 592 19 572 294 391 62 0
Route 9	0 121 51 150 16 453 115 199 411 514 443 171 516 0
Route 10	0 124 141 266 555 151 478 73 147 356 57 352 583 0
Route 11	0 130 213 12 81 198 263 343 357 52 465 290 552 0
Route 12	0 148 279 311 97 328 79 248 217 192 405 119 591 0
Route 13	0 153 83 132 307 574 74 15 65 170 212 487 0
Route 14	0 154 475 358 50 36 474 146 188 0
Route 15	0 163 365 298 42 205 315 519 112 2 21 0
Route 16	0 173 228 376 500 421 564 381 350 392 312 218 0
Route 17	0 184 254 361 402 470 558 515 200 123 335 455 0
Route 18	0 236 538 116 152 512 162 375 543 0
Route 19	0 276 429 143 559 301 245 364 504 195 422 430 0
Route 20	0 280 277 554 541 157 306 327 456 462 438 325 432 0
Route 21	0 284 118 329 452 568 393 181 308 338 13 0
Route 22	0 287 225 528 32 428 485 136 489 288 539 135 506 0

Route 23	0 289 579 252 367 250 99 320 598 0
Route 24	0 296 397 168 114 450 495 133 3 349 571 420 501 28 0
Route 25	0 303 399 111 566 560 167 258 510 256 55 224 0
Route 26	0 316 546 261 380 337 96 362 37 318 242 398 234 0
Route 27	0 322 319 424 548 499 106 286 206 551 267 355 0
Route 28	0 330 175 222 476 194 179 180 4 496 138 523 0
Route 29	0 334 56 529 562 33 127 282 75 158 0
Route 30	0 336 230 237 509 388 299 406 447 187 419 0
Route 31	0 345 251 340 464 197 244 120 221 59 0
Route 32	0 348 64 66 215 293 285 58 300 88 468 0
Route 33	0 353 469 144 449 326 513 498 134 274 460 0
Route 34	0 368 366 305 573 371 243 339 544 553 444 536 442 0
Route 35	0 390 178 23 354 517 384 492 360 90 0
Route 36	0 404 532 268 434 196 257 67 372 491 584 396 295 229 0
Route 37	0 407 479 44 262 190 176 359 69 409 8 0
Route 38	0 410 387 569 441 246 232 534 164 304 125 70 0
Route 39	0 416 445 185 227 241 98 104 595 518 233 0
Route 40	0 435 437 505 91 378 101 259 297 0
Route 41	0 436 346 521 590 117 527 373 165 93 0
Route 42	0 446 585 369 145 161 401 95 94 22 385 6 386 0
Route 43	0 458 379 459 545 207 323 48 561 159 503 0
Route 44	0 467 486 216 433 578 41 25 174 292 29 204 589 103 0
Route 45	0 493 408 363 253 172 550 377 271 321 0
Route 46	0 511 235 177 137 17 425 189 202 166 10 0
Route 47	0 520 488 240 105 82 249 494 275 139 497 210 0
Route 48	0 525 40 415 342 412 463 45 580 155 0
Route 49	0 531 317 110 265 63 537 383 149 31 9 557 0
Route 50	0 533 374 128 7 278 427 477 599 483 182 403 108 160 0
Route 51	0 547 507 524 203 480 522 273 283 347 0
Route 52	0 549 54 209 302 556 211 309 247 60 417 47 0
Route 53	0 570 314 140 281 231 78 526 310 565 482 535 264 461 0
Route 54	0 576 238 77 582 71 131 34 0
Route 55	0 577 344 142 457 226 14 27 581 451 600 0
Route 56	0 587 440 413 5 53 448 39 530 567 270 191 0

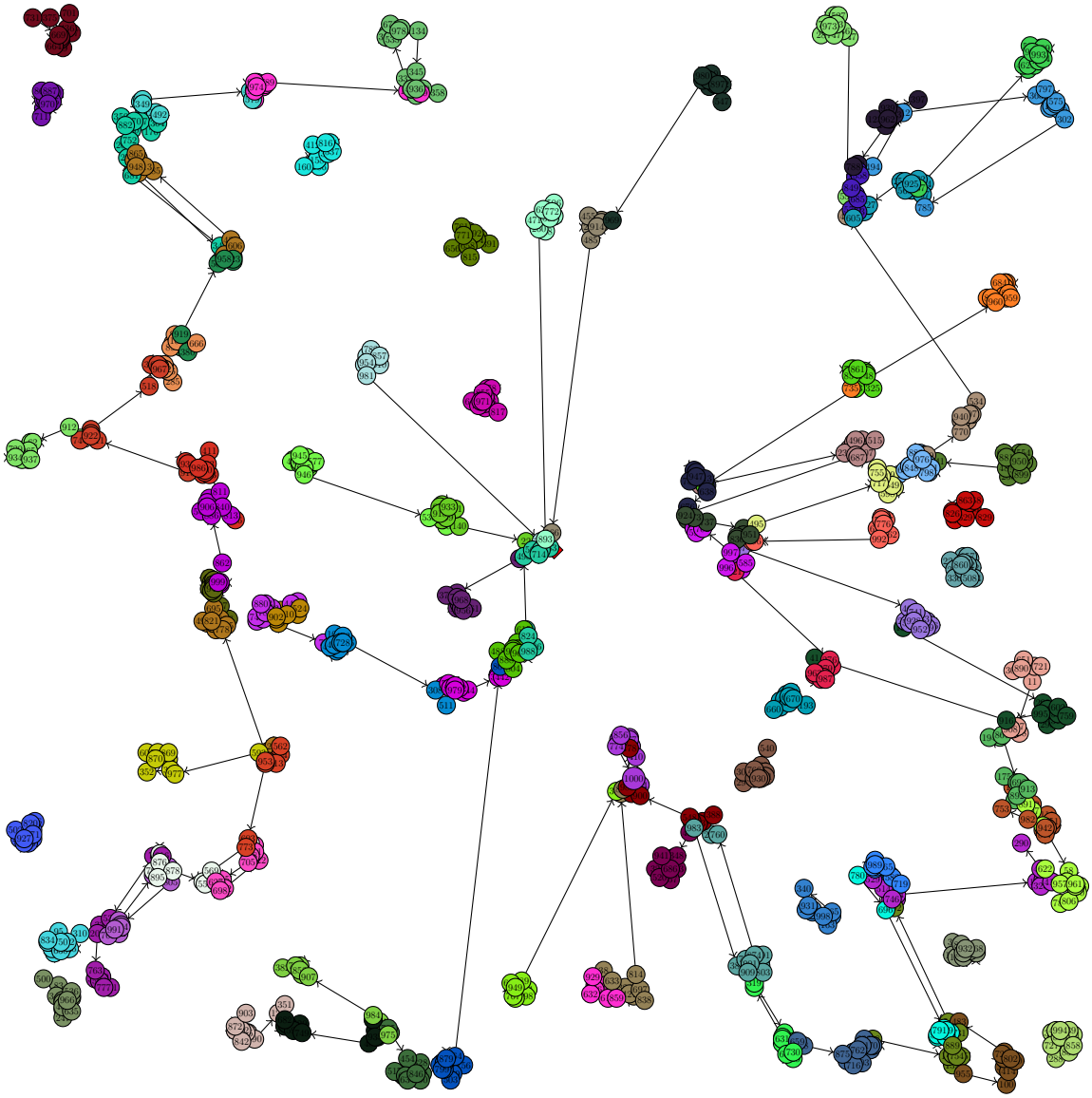
Instanz C1_800



Knotenanzahl Maximalkapazität Distanz Routenanzahl	801 200 23690,76 72
Route 1	0 12 211 98 625 571 640 39 588 621 719 200 0
Route 2	0 18 734 459 416 738 59 783 191 167 266 0
Route 3	0 34 325 118 742 355 684 784 678 196 670 601 600 234 0
Route 4	0 47 758 181 465 113 57 727 302 698 581 0
Route 5	0 49 162 93 441 164 549 422 708 513 345 185 0
Route 6	0 68 434 341 128 659 396 247 76 563 574 0
Route 7	0 96 527 446 195 259 533 733 768 623 464 67 176 0
Route 8	0 101 596 524 89 587 338 716 123 546 115 130 228 0
Route 9	0 105 697 404 84 317 455 735 218 51 40 83 217 288 0
Route 10	0 111 666 705 291 384 619 268 603 793 214 671 669 0
Route 11	0 133 319 451 449 70 694 667 332 628 447 638 0
Route 12	0 145 620 470 131 139 359 552 643 406 275 777 0
Route 13	0 153 626 741 81 729 95 577 79 0
Route 14	0 165 306 597 467 595 612 771 429 334 0
Route 15	0 166 137 136 502 677 417 584 410 356 23 438 15 0
Route 16	0 168 490 474 103 303 378 213 358 380 230 537 297 0
Route 17	0 174 252 711 178 745 781 663 180 285 471 270 714 535 0
Route 18	0 177 44 3 161 35 348 309 112 365 739 171 227 468 0

Route 19	0 192 138 645 120 789 506 639 550 730 554 431 460 0
Route 20	0 220 382 791 55 462 16 368 740 755 282 94 686 0
Route 21	0 221 10 318 609 720 558 576 736 593 0
Route 22	0 226 64 301 373 399 770 351 313 436 635 262 0
Route 23	0 229 756 536 661 700 295 510 579 24 344 0
Route 24	0 238 194 54 189 354 405 31 90 127 442 627 33 0
Route 25	0 255 439 545 543 312 271 330 119 172 48 316 674 403 0
Route 26	0 260 599 205 241 611 767 71 696 528 641 583 0
Route 27	0 269 726 279 106 553 246 676 773 186 0
Route 28	0 277 377 235 800 702 518 114 240 360 425 146 392 0
Route 29	0 296 154 567 785 216 310 375 320 522 590 452 0
Route 30	0 304 458 144 688 507 765 328 718 335 369 0
Route 31	0 308 764 662 116 77 787 747 633 499 443 580 0
Route 32	0 314 9 586 263 769 614 349 453 402 560 0
Route 33	0 329 210 591 26 236 428 80 274 423 290 762 511 394 0
Route 34	0 352 743 257 264 616 491 798 339 0
Route 35	0 363 699 766 175 475 179 433 184 557 66 629 0
Route 36	0 366 124 379 46 772 232 636 237 794 199 0
Route 37	0 400 278 99 37 357 188 607 20 737 91 0
Route 38	0 407 411 754 242 61 158 408 29 386 398 763 74 615 0
Route 39	0 456 797 505 122 514 75 190 526 294 56 0
Route 40	0 461 117 397 675 788 327 331 219 324 156 361 385 372 0
Route 41	0 463 757 517 435 775 665 65 486 542 121 248 631 0
Route 42	0 473 480 337 679 613 782 562 245 155 673 340 485 540 0
Route 43	0 488 8 482 197 712 58 445 564 682 50 261 182 617 0
Route 44	0 492 672 163 265 521 568 725 244 204 722 0
Route 45	0 497 420 642 477 654 149 390 142 413 751 432 0
Route 46	0 500 376 202 685 85 704 207 42 201 343 364 0
Route 47	0 501 132 608 280 752 143 692 478 605 476 52 0
Route 48	0 503 573 225 1 108 746 680 512 286 721 0
Route 49	0 525 472 187 786 134 448 11 287 86 104 717 419 27 129 0
Route 50	0 544 126 594 305 483 110 281 566 183 529 534 0
Route 51	0 548 508 681 637 53 293 539 555 648 272 724 148 520 0
Route 52	0 556 109 321 173 198 572 283 157 326 224 7 0
Route 53	0 565 489 701 418 276 519 253 532 311 22 78 644 0
Route 54	0 569 193 653 570 748 651 147 732 706 336 585 0
Route 55	0 578 575 715 561 233 323 647 169 206 504 618 0
Route 56	0 592 723 538 256 100 43 523 481 602 159 38 0
Route 57	0 610 424 776 516 622 391 450 258 0
Route 58	0 634 97 728 82 731 454 292 383 531 509 342 493 0
Route 59	0 646 683 170 102 693 203 415 606 5 347 0
Route 60	0 649 6 691 774 778 2 223 299 370 222 0
Route 61	0 652 273 69 289 73 624 412 346 14 92 19 371 0
Route 62	0 656 498 792 307 30 333 744 664 630 0
Route 63	0 657 267 212 759 21 437 495 632 374 63 0
Route 64	0 658 487 496 208 350 315 88 753 494 393 604 0
Route 65	0 668 381 703 254 479 457 125 17 790 695 0
Route 66	0 689 795 799 582 32 107 300 209 469 530 353 0
Route 67	0 690 41 796 421 87 598 655 484 215 322 547 135 0
Route 68	0 709 414 589 707 4 395 660 389 687 761 45 249 0
Route 69	0 710 430 409 250 284 388 140 151 541 387 160 780 650 0
Route 70	0 713 25 150 243 141 298 749 231 62 0
Route 71	0 760 239 444 401 551 36 466 362 559 72 28 440 0
Route 72	0 779 152 750 251 515 426 367 60 427 13 0

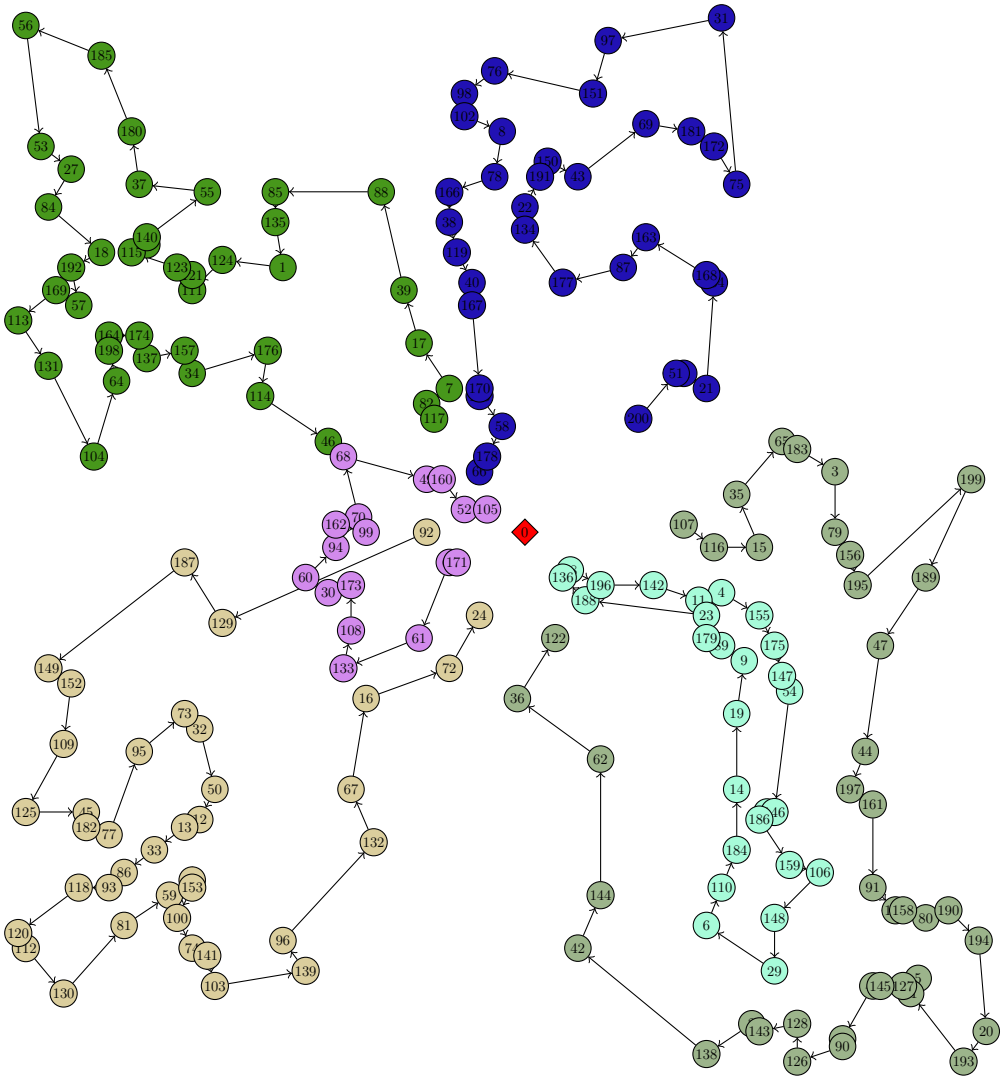
Instanz C1_1000



Knotenanzahl	1001
Maximalkapazität	200
Distanz	39364,68
Routenanzahl	90
Route 1	0 23 610 119 298 543 902 282 40 524 0
Route 2	0 28 775 973 188 527 245 793 747 636 147 552 0
Route 3	0 38 633 212 324 421 369 261 838 697 814 30 884 0
Route 4	0 47 461 678 316 705 415 322 627 698 462 122 0
Route 5	0 61 912 62 53 265 2 126 720 934 102 407 181 937 0
Route 6	0 66 297 723 920 227 943 850 828 161 71 567 433 0
Route 7	0 87 469 741 733 396 446 619 952 15 73 928 542 293 0
Route 8	0 106 158 81 385 537 124 816 412 160 0
Route 9	0 108 184 139 14 645 754 700 889 531 54 292 0
Route 10	0 123 939 397 440 743 486 962 115 528 788 0
Route 11	0 144 539 111 551 94 843 942 877 609 982 753 0
Route 12	0 146 171 270 832 827 370 154 868 932 825 35 0
Route 13	0 167 299 449 602 546 759 377 296 995 916 41 0
Route 14	0 194 84 867 373 692 133 913 892 507 175 0
Route 15	0 208 936 425 332 535 303 226 671 978 851 134 345 409 358 0
Route 16	0 238 597 287 772 506 626 471 260 893 0
Route 17	0 239 770 611 940 314 807 738 534 451 0
Route 18	0 246 819 999 862 886 174 522 906 541 586 463 811 840 813 0
Route 19	0 286 988 277 824 355 498 561 714 663 36 183 0
Route 20	0 291 148 57 131 778 129 150 821 8 491 695 383 545 0
Route 21	0 309 890 651 253 721 11 847 468 595 908 779 0

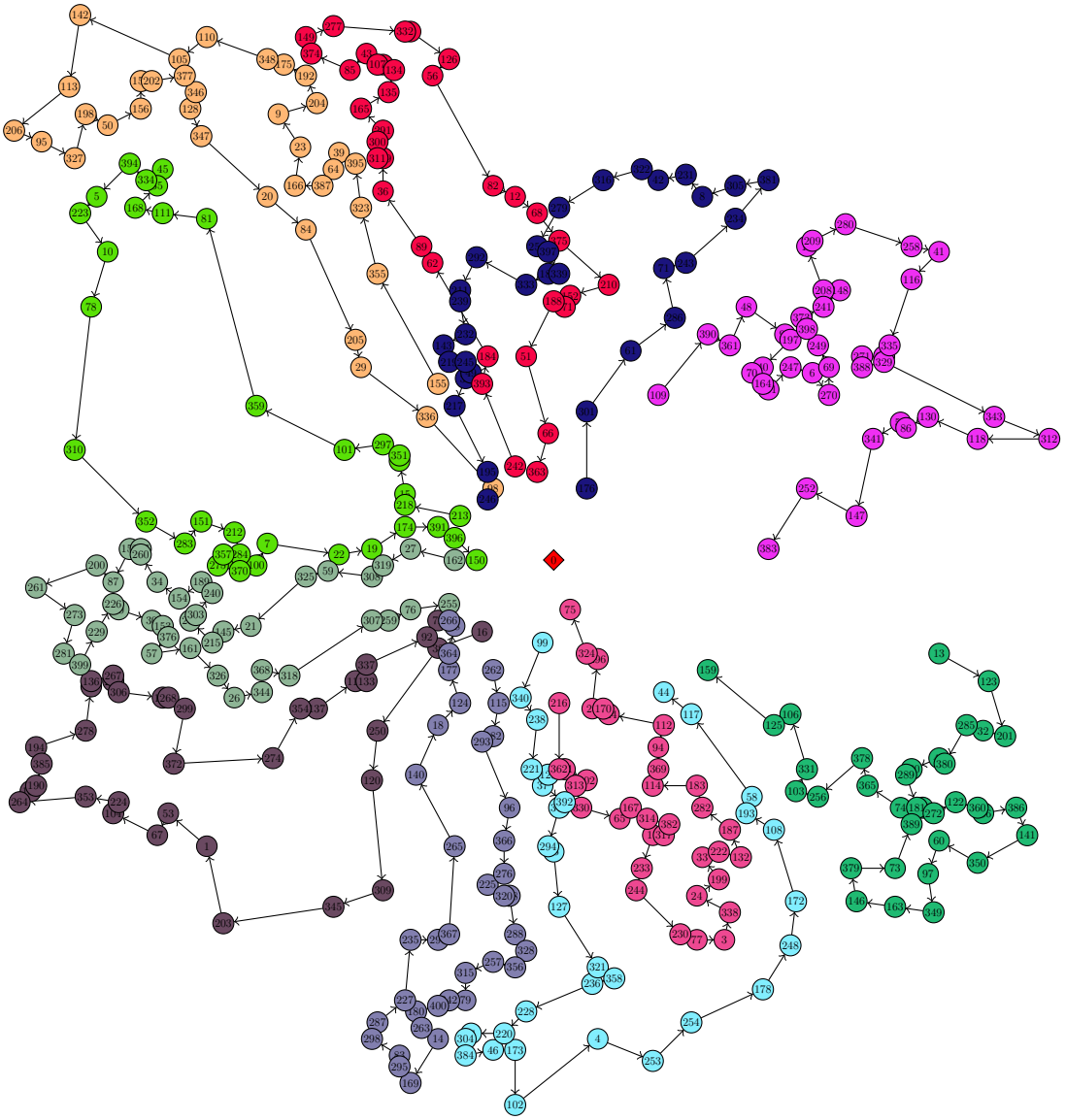
Route 22	0 310 642 750 264 236 80 630 215 271 381 557 834 95 0
Route 23	0 319 42 110 730 229 639 333 467 631 76 145 0
Route 24	0 336 508 254 152 392 101 112 157 29 757 860 453 283 221 0
Route 25	0 337 231 1 70 496 166 515 307 687 617 576 0
Route 26	0 364 176 237 570 350 882 400 752 269 275 681 341 317 0
Route 27	0 380 44 9 919 505 142 958 258 823 652 0
Route 28	0 387 436 701 375 731 669 16 490 489 49 664 79 0
Route 29	0 388 321 353 405 548 900 136 985 478 0
Route 30	0 394 441 493 703 880 781 742 604 679 77 0
Route 31	0 411 563 818 938 986 680 189 27 153 116 0
Route 32	0 435 250 629 829 523 117 768 657 863 330 248 826 0
Route 33	0 438 218 371 745 968 643 143 844 956 661 0
Route 34	0 447 994 608 839 361 858 835 288 727 432 853 384 0
Route 35	0 473 109 424 417 90 395 65 519 728 186 308 511 0
Route 36	0 481 313 876 795 114 895 416 284 878 172 559 569 389 0
Route 37	0 488 888 715 804 964 926 923 808 667 615 0
Route 38	0 492 219 170 178 349 216 240 401 579 0
Route 39	0 494 512 363 797 45 68 575 295 187 19 180 302 785 0
Route 40	0 495 717 273 598 556 649 99 704 755 599 0
Route 41	0 499 348 327 243 686 357 520 32 941 0
Route 42	0 500 83 347 367 448 452 113 247 635 966 121 257 536 0
Route 43	0 510 857 256 782 39 404 954 125 981 311 0
Route 44	0 521 276 31 279 466 987 801 726 963 0
Route 45	0 529 514 205 746 37 204 164 323 242 419 179 450 290 0
Route 46	0 538 944 422 974 972 331 103 553 789 149 739 0
Route 47	0 540 169 230 241 756 930 343 766 504 294 306 0
Route 48	0 547 202 897 118 6 574 210 980 268 969 0
Route 49	0 562 374 513 159 418 953 560 603 773 0
Route 50	0 568 217 24 267 104 132 925 526 223 427 206 605 0
Route 51	0 577 533 329 249 945 20 46 689 138 905 225 946 530 59 0
Route 52	0 581 933 52 368 654 335 918 444 699 140 22 0
Route 53	0 593 977 390 352 600 870 786 274 869 0
Route 54	0 606 431 185 713 75 865 378 96 91 948 571 516 0
Route 55	0 616 74 532 33 209 763 360 320 564 777 301 0
Route 56	0 623 734 228 107 776 162 992 796 694 0
Route 57	0 648 949 197 767 214 198 64 589 305 0
Route 58	0 650 881 233 554 950 423 85 899 289 572 255 487 141 0
Route 59	0 659 88 875 706 716 203 693 762 601 93 182 470 232 0
Route 60	0 660 3 402 456 565 193 670 646 263 207 0
Route 61	0 662 674 871 196 465 509 927 222 502 820 406 0
Route 62	0 666 4 17 105 97 18 89 393 472 356 285 0
Route 63	0 712 191 898 915 979 732 278 544 359 211 442 0
Route 64	0 735 960 810 620 959 841 580 304 480 684 833 56 0
Route 65	0 751 439 588 947 613 72 638 676 0
Route 66	0 765 434 454 517 445 201 624 634 846 200 92 328 0
Route 67	0 780 696 195 769 177 607 135 724 379 525 791 0
Route 68	0 805 672 894 372 709 991 641 338 234 0
Route 69	0 815 658 656 771 761 168 592 464 281 391 0
Route 70	0 817 376 78 13 855 794 971 904 612 725 628 130 0
Route 71	0 837 43 280 790 864 993 809 60 584 990 625 482 0
Route 72	0 845 614 812 690 408 887 866 413 729 970 403 711 0
Route 73	0 849 476 342 555 354 558 549 685 67 7 443 346 578 0
Route 74	0 854 783 640 861 339 457 748 325 673 573 0
Route 75	0 856 758 774 1000 25 644 410 5 477 398 0
Route 76	0 879 501 683 344 426 799 594 82 98 503 399 156 764 800 0
Route 77	0 883 55 220 976 458 591 51 798 637 848 655 647 10 0
Route 78	0 891 707 885 58 587 961 831 318 710 806 787 957 622 0
Route 79	0 896 497 484 935 48 749 429 596 235 475 682 0
Route 80	0 903 872 830 459 822 675 50 842 190 127 351 0
Route 81	0 910 911 744 922 736 34 518 315 688 474 967 0
Route 82	0 914 362 621 213 251 455 252 86 485 266 0
Route 83	0 921 677 479 437 719 582 300 965 740 989 917 0
Route 84	0 924 155 873 199 737 420 951 566 21 69 583 836 0
Route 85	0 929 12 192 366 632 618 859 63 262 0
Route 86	0 931 173 151 137 998 163 26 665 340 0
Route 87	0 955 100 414 334 430 802 326 718 722 550 483 0
Route 88	0 975 653 244 984 907 428 852 708 460 784 382 0
Route 89	0 983 386 792 901 909 259 803 165 128 691 668 874 760 272 0
Route 90	0 996 120 365 585 997 702 312 224 590 0

Instanz C2_200



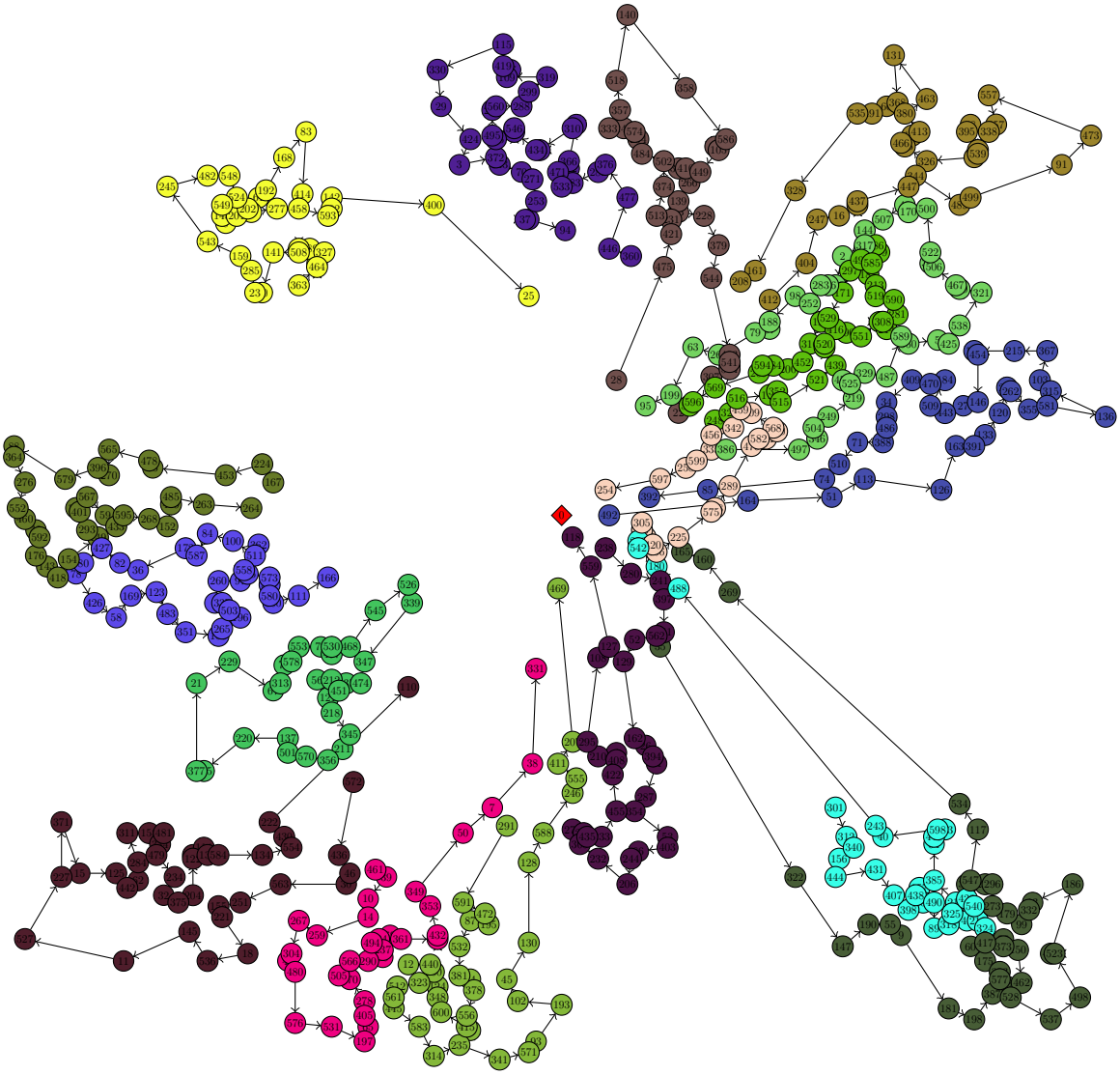
Knotenanzahl Maximalkapazität Distanz Routenanzahl	201 700 1482,43 6
Route 1	0 28 196 142 11 4 155 175 147 54 146 83 186 159 106 148 29 6 110 184 14 19 9 89 179 23 188 136 0
Route 2	0 92 129 187 149 152 109 125 45 182 77 95 73 32 50 12 13 33 86 93 118 120 112 130 81 59 10 153 100 74 141 103 139 96 132 67 16 72 24 0
Route 3	0 107 116 15 35 65 183 3 79 156 195 199 189 47 44 197 161 91 101 158 80 190 194 20 193 71 5 127 145 48 25 90 126 128 143 2 138 42 144 62 36 122 0
Route 4	0 117 82 7 17 39 88 85 135 1 124 111 121 123 115 26 140 55 37 180 185 56 53 27 84 18 192 57 169 113 131 104 64 198 164 174 137 157 34 176 114 46 0
Route 5	0 171 63 61 133 108 173 30 60 94 162 99 70 68 49 160 52 105 0
Route 6	0 200 51 41 21 154 168 163 87 177 134 22 191 150 43 69 181 172 75 31 97 151 76 98 102 8 78 166 38 119 40 167 170 165 58 178 66 0

Instanz C2_400



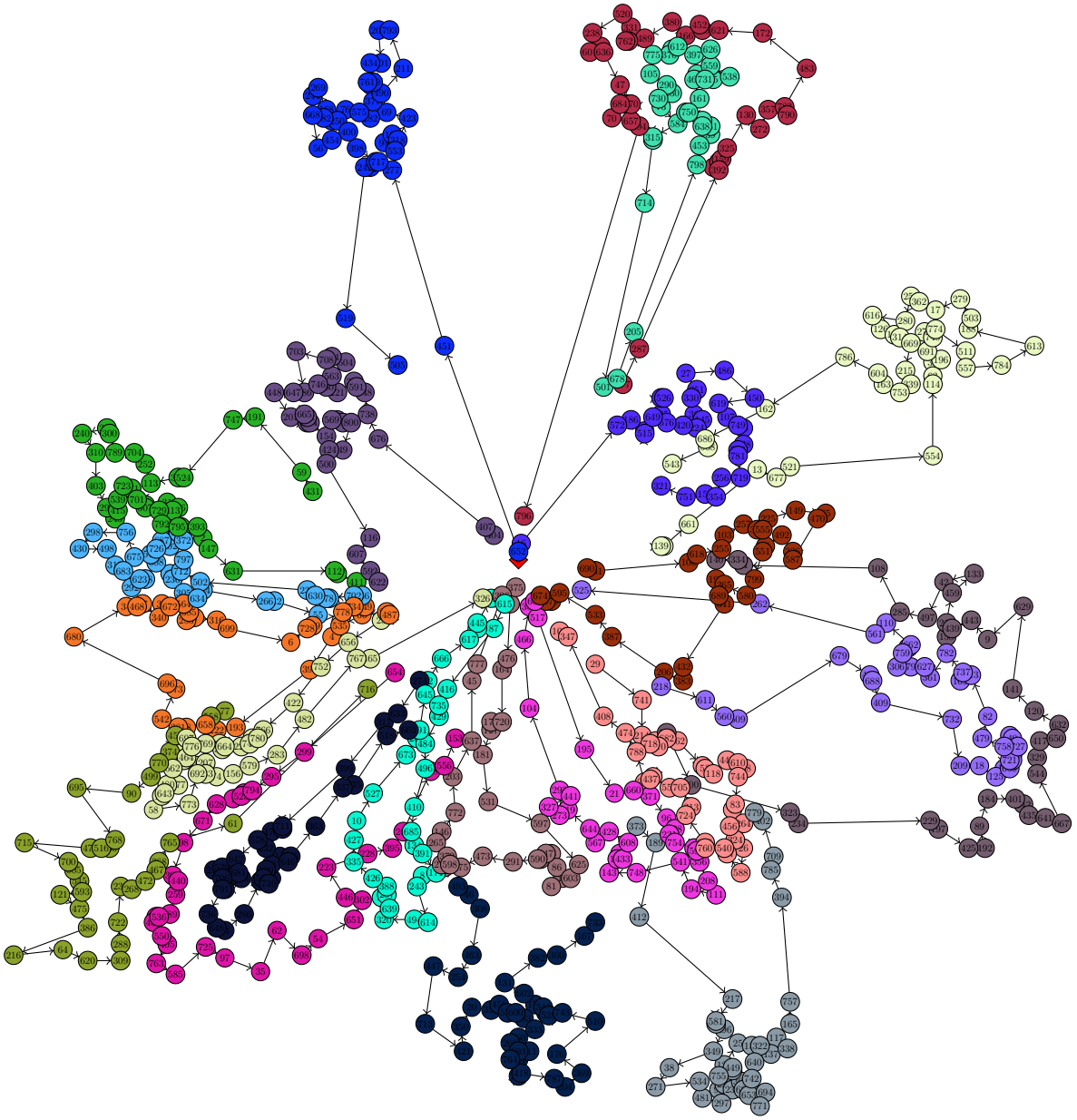
Knotenanzahl	401
Maximalkapazität	700
Distanz	3123.99
Routenanzahl	11
Route 1	0 13 123 201 32 285 380 47 160 289 181 121 272 122 360 186 386 141 350 60 97 349 163 146 379 73 389 74 365 378 256 103 331 106 125 159 0
Route 2	0 16 38 250 120 309 345 203 1 53 67 104 224 353 264 139 190 385 194 278 54 136 267 214 306 196 268 299 372 274 354 137 119 133 337 92 72 0
Route 3	0 99 340 238 221 129 37 392 371 294 35 127 321 358 236 228 220 91 304 384 46 173 102 4 253 254 178 248 172 108 193 58 117 44 0
Route 4	0 109 390 361 48 93 197 40 70 164 131 247 6 270 69 249 398 373 241 148 208 11 209 280 258 41 116 335 269 271 388 329 343 312 118 130 86 52 341 147 252 383 0
Route 5	0 155 355 323 395 39 64 387 166 23 9 204 192 175 348 110 105 142 113 206 95 327 198 50 156 157 202 377 346 128 347 20 84 205 29 336 98 0
Route 6	0 162 27 319 308 59 325 21 145 215 207 303 240 189 154 34 260 237 158 87 200 261 273 281 399 229 226 80 30 153 376 57 161 326 26 344 368 318 307 259 76 255 0
Route 7	0 176 301 61 286 71 243 234 381 305 8 231 42 322 316 279 251 397 339 185 333 292 211 239 232 143 219 245 49 31 217 195 246 0
Route 8	0 213 218 15 28 351 297 101 359 81 111 168 55 45 334 394 5 223 10 78 310 352 283 151 212 284 2 357 275 370 100 7 22 19 174 391 396 150 0
Route 9	0 216 362 191 302 313 330 65 167 314 382 317 17 233 244 230 77 3 338 24 199 33 222 132 187 282 183 114 369 94 112 144 170 25 296 324 75 0
Route 10	0 242 393 184 62 89 36 179 311 300 291 165 135 134 63 107 43 85 374 149 277 332 90 126 56 82 12 68 375 210 152 171 188 51 66 363 0
Route 11	0 262 115 182 293 96 366 276 225 320 138 288 328 356 257 315 79 342 400 180 263 14 169 295 83 298 287 227 235 290 367 265 140 18 124 177 364 88 266 0

Instanz C2_600



Knotenanzahl	601
Maximalkapazität	700
Distanz	6269,95
Routenanzahl	17
Route 1	0 28 475 421 231 513 374 502 484 294 574 240 333 357 518 140 358 586 92 105 449 410 389 266 139 33 228 379 544 493 541 223 514 307 22 0
Route 2	0 35 322 147 190 55 9 181 198 387 577 393 175 60 417 96 373 150 462 17 528 537 498 523 402 186 13 332 99 179 273 296 272 382 547 117 534 269 160 165 0
Route 3	0 167 224 453 75 478 565 270 396 579 88 364 276 552 460 42 592 176 143 418 154 149 293 401 119 567 59 433 595 268 152 429 485 263 264 0
Route 4	0 238 280 241 397 441 562 52 129 162 26 394 31 287 354 53 403 86 244 206 232 303 275 292 435 233 455 422 408 4 210 295 108 127 559 118 0
Route 5	0 248 320 516 116 352 515 521 439 369 520 416 390 551 77 308 214 281 590 519 213 187 585 279 286 496 297 171 529 191 316 452 200 384 594 209 569 596 564 0
Route 6	0 291 591 472 195 87 532 381 44 378 556 76 415 600 348 124 250 440 12 323 512 561 445 583 314 235 341 571 93 193 102 45 130 128 588 246 555 411 207 469 0
Route 7	0 301 312 340 156 444 431 407 398 438 157 1 302 490 89 318 325 41 216 324 174 540 423 217 385 64 448 173 598 40 243 488 180 542 203 0
Route 8	0 305 177 107 420 306 225 575 428 289 476 582 151 350 568 309 459 255 342 230 456 335 599 258 597 254 0
Route 9	0 339 347 474 135 451 212 56 121 218 345 211 356 570 501 137 220 365 377 21 229 67 313 73 578 553 72 530 457 468 545 526 0
Route 10	0 360 446 477 376 282 383 533 471 132 366 54 310 274 434 336 546 465 495 236 560 288 299 319 109 47 419 115 330 29 424 3 372 343 70 271 253 37 19 94 0
Route 11	0 362 511 100 84 587 172 36 82 427 80 78 426 58 169 123 483 351 153 265 196 503 337 6 260 97 558 205 573 517 580 300 111 166 0
Route 12	0 363 464 24 327 189 242 508 141 20 23 285 159 543 245 482 548 524 549 148 8 201 202 90 277 61 192 168 83 414 458 593 182 142 400 25 0
Route 13	0 386 497 346 504 249 219 525 450 329 487 589 550 5 425 538 321 399 467 506 359 522 500 170 43 507 144 317 2 106 226 283 252 98 188 57 79 261 63 199 95 0
Route 14	0 412 404 247 16 239 437 447 344 489 499 91 473 557 257 338 48 395 104 539 81 326 185 466 183 413 380 463 131 368 66 491 535 328 161 208 0
Route 15	0 461 39 10 14 259 267 304 49 480 576 531 197 65 405 278 370 505 566 290 68 237 178 494 194 69 361 406 432 353 349 50 7 38 331 0
Route 16	0 492 164 51 113 126 163 391 133 120 256 262 355 581 136 315 103 367 215 112 454 146 27 443 509 184 470 409 34 298 486 388 71 510 74 85 392 0
Route 17	0 572 436 46 30 563 251 155 221 18 536 145 11 527 227 371 15 125 442 62 284 311 158 481 334 479 234 32 375 204 122 101 114 138 584 134 554 430 222 110 0

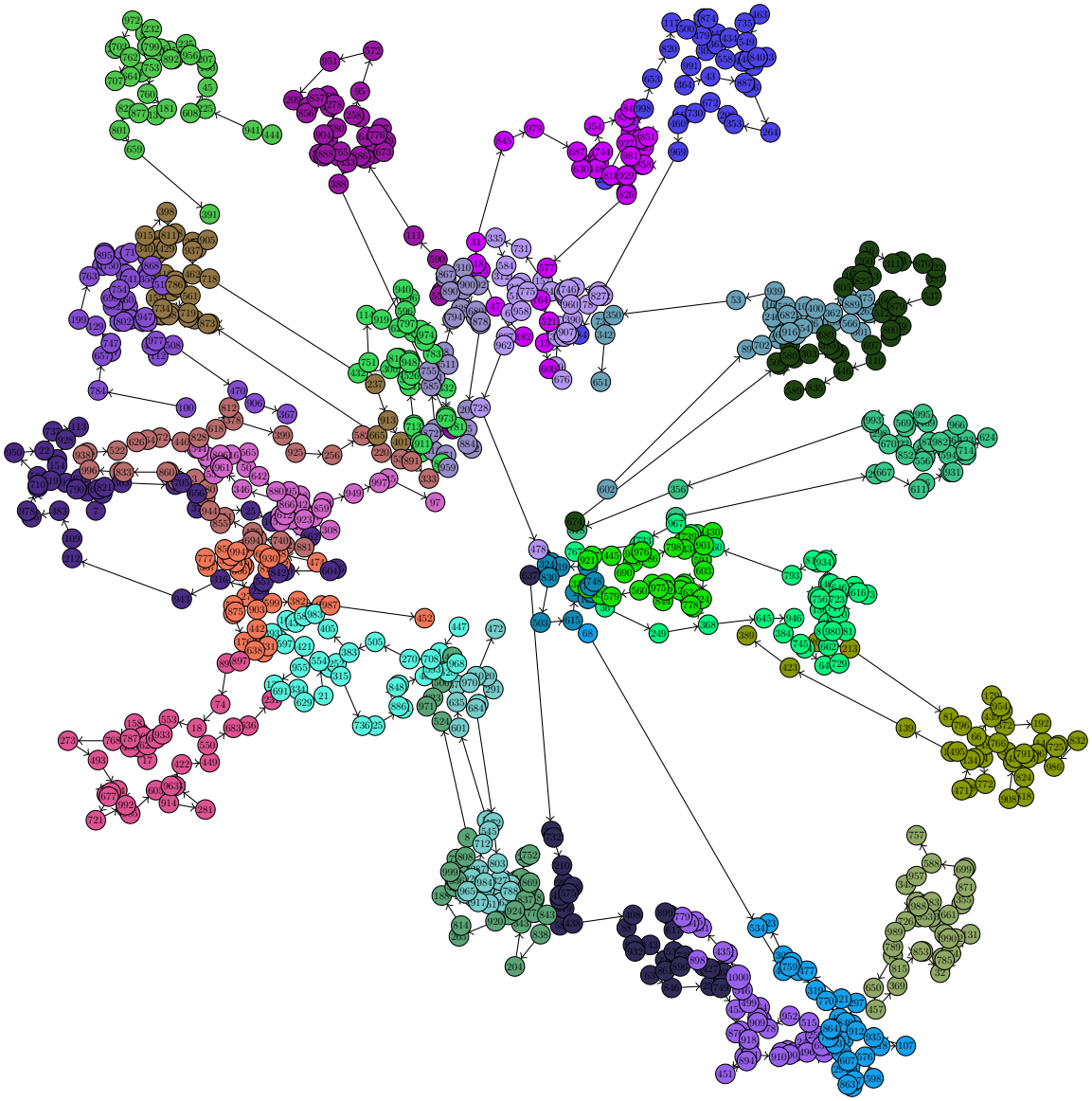
Instanz C2_800



Knotenanzahl Maximalkapazität Distanz Routenanzahl	801 700 9601,51 22
Route 1	0 16 572 186 515 92 649 246 526 135 576 420 224 145 36 330 251 27 486 450 619 107 749 68 438 570 781 719 256 354 152 751 321 0
Route 2	0 23 656 752 422 766 780 436 745 220 664 301 284 769 776 697 276 464 248 80 562 177 480 643 199 58 773 692 170 207 583 74 156 579 283 482 767 465 326 0
Route 3	0 29 741 214 718 226 710 682 462 50 53 118 442 610 88 744 83 5 264 456 324 26 588 367 540 760 355 724 313 705 552 91 437 241 788 474 408 347 109 0
Route 4	0 77 558 457 374 132 770 499 90 695 768 136 516 471 715 700 605 545 593 121 475 386 216 64 620 309 288 722 239 268 472 467 368 765 61 716 0
Route 5	0 139 75 661 13 677 521 554 114 63 33 196 691 740 270 774 511 557 784 613 188 503 279 17 362 258 280 616 126 129 314 669 215 339 753 163 604 786 162 686 568 543 0
Route 6	0 218 611 560 509 679 455 688 409 732 209 18 125 72 659 721 707 495 727 40 30 758 479 82 513 169 737 782 361 627 79 306 759 461 662 110 561 262 525 0
Route 7	0 274 633 317 399 711 512 174 232 578 642 681 506 574 529 213 706 606 791 736 648 44 286 532 565 739 41 180 93 14 646 485 363 337 235 518 663 0
Route 8	0 373 189 412 217 581 546 296 349 38 271 534 481 297 99 755 227 233 624 653 187 771 694 742 303 419 449 123 25 157 322 640 137 338 117 165 757 394 785 709 402 779 0
Route 9	0 375 51 476 164 720 179 127 181 531 597 625 603 19 81 86 28 547 537 590 291 473 275 598 148 210 379 265 146 772 203 637 45 777 396 0

Route 10	0 404 407 676 738 548 591 376 221 460 746 563 504 514 708 703 586 647 448 201 665 522 364 444 154 569 48 414 800 49 424 500 116 607 592 622 0
Route 11	0 431 59 191 747 524 332 113 252 704 789 300 261 240 310 403 294 245 415 539 723 571 701 384 507 729 31 346 413 792 795 343 393 2 147 631 112 60 411 0
Route 12	0 477 287 392 1 22 159 325 130 272 357 790 783 483 172 621 366 452 166 380 489 609 762 331 520 238 601 198 636 47 670 589 684 70 657 594 796 0
Route 13	0 487 281 66 535 4 390 193 222 658 78 381 542 173 696 680 34 468 67 340 37 672 564 385 253 3 316 699 6 728 359 778 734 549 0
Route 14	0 493 46 308 463 254 447 713 421 350 20 353 458 573 600 183 333 160 260 693 311 124 764 278 418 787 204 369 176 510 743 528 328 344 178 602 131 382 360 167 733 0
Route 15	0 530 100 323 234 229 197 425 192 89 184 401 342 435 641 667 544 329 417 650 632 120 141 629 9 443 439 158 212 459 133 42 497 285 108 71 334 140 0
Route 16	0 566 702 185 378 630 237 502 712 797 372 43 138 12 102 687 726 577 675 756 298 430 498 319 683 292 623 478 508 32 236 305 634 144 266 142 55 0
Route 17	0 615 87 416 735 429 484 496 410 685 134 391 171 8 243 614 494 320 639 11 267 388 426 335 427 10 527 673 202 491 645 52 666 617 247 445 24 0
Route 18	0 652 451 277 553 94 122 318 423 69 490 211 793 200 101 434 761 599 57 377 282 575 76 250 115 582 168 269 244 668 56 454 400 398 289 717 84 242 519 505 0
Route 19	0 654 299 295 794 488 523 7 628 671 98 304 440 259 389 536 150 406 550 405 763 585 725 97 35 62 698 54 651 302 446 223 228 395 263 358 556 153 0
Route 20	0 655 517 195 21 660 371 96 128 231 754 219 596 356 208 111 194 541 748 143 175 433 155 608 428 567 644 119 273 327 441 293 104 466 352 0
Route 21	0 678 205 798 453 39 151 638 348 750 635 161 469 85 731 15 538 559 626 397 612 182 370 775 105 290 230 730 73 584 315 312 714 501 0
Route 22	0 690 351 106 618 255 103 257 345 555 225 149 65 470 587 336 492 95 551 799 580 365 190 689 341 432 383 206 387 533 595 249 307 674 0

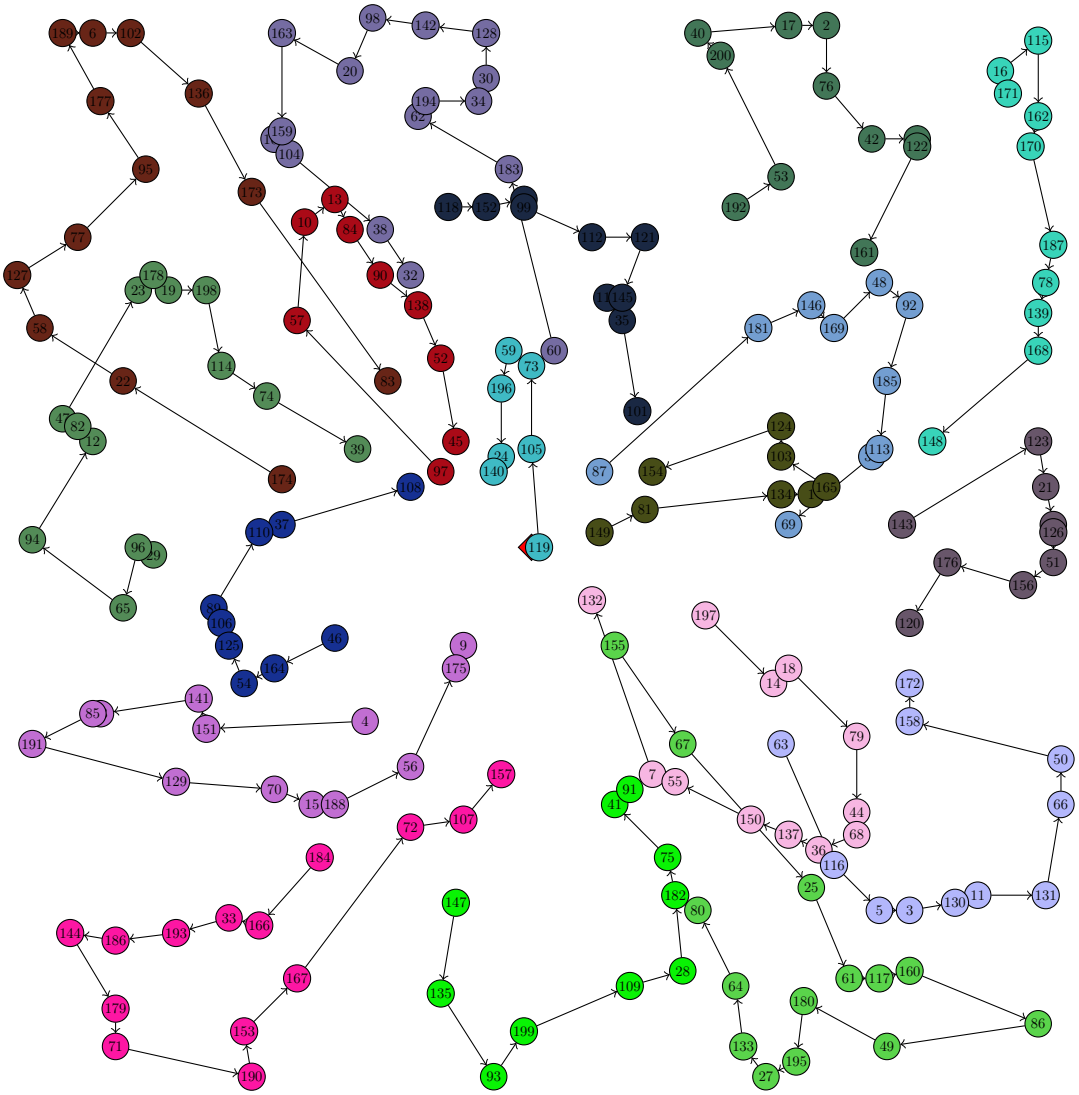
Instanz C2_1000



Knotenanzahl Maximalkapazität Distanz Routenanzahl	1001 700 14258,22 28
Route 1	0 36 249 368 645 946 384 745 654 64 729 662 813 980 581 570 487 69 321 136 543 756 717 723 616 173 75 646 402 349 163 934 276 819 793 660 417 967 733 271 767 0
Route 2	0 68 534 409 759 370 319 770 404 715 864 706 693 381 510 607 223 294 863 165 727 598 576 518 107 935 912 443 849 297 621 477 490 302 323 0
Route 3	0 100 784 657 377 747 129 199 763 167 750 307 895 251 24 71 51 868 198 517 358 741 426 754 467 695 502 360 87 33 802 418 464 947 841 977 458 112 508 470 906 367 0
Route 4	0 182 47 115 118 31 845 979 687 630 448 218 305 227 744 354 834 847 200 326 692 42 927 782 851 614 625 9 981 858 338 929 810 279 826 577 764 521 15 600 0
Route 5	0 238 998 653 820 117 500 479 178 874 242 434 735 463 549 840 133 120 489 456 558 361 175 301 991 364 43 887 366 264 353 208 672 730 419 460 969 184 0
Route 6	0 267 213 81 796 66 439 179 954 410 372 192 144 725 65 832 986 396 791 583 824 818 908 171 485 30 766 48 34 772 468 471 134 49 495 155 139 423 389 0
Route 7	0 285 604 150 842 557 228 316 943 212 109 483 156 978 304 710 39 568 950 22 737 113 928 6 154 619 92 632 790 193 7 606 622 821 363 55 705 656 37 25 101 862 0
Route 8	0 308 923 196 344 571 425 612 90 866 290 346 961 839 552 116 214 161 544 806 716 565 50 642 880 795 548 742 191 373 14 587 859 189 949 997 945 97 0
Route 9	0 324 215 830 503 615 2 153 527 709 748 280 119 98 0
Route 10	0 420 291 539 684 172 803 327 274 91 268 788 663 380 761 284 984 216 917 140 831 965 247 688 926 287 712 545 12 601 635 376 970 610 472 0
Route 11	0 444 941 225 608 45 160 207 956 235 10 892 145 226 263 799 125 83 232 972 703 288 762 707 664 217 753 760 181 137 877 822 801 659 391 0

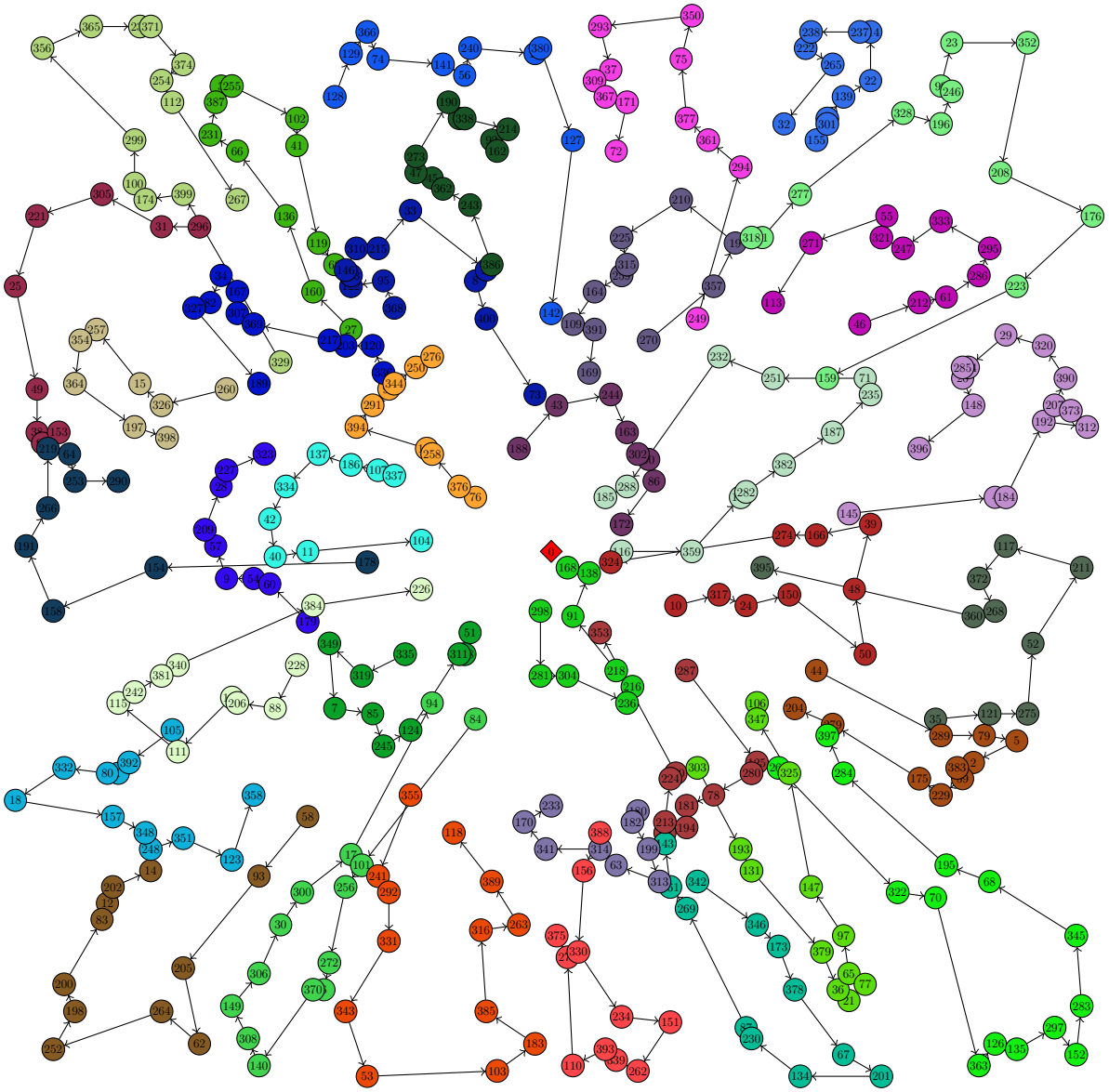
Route 12	0 447 414 708 270 505 383 405 983 586 431 108 393 104 597 421 955 135 691 334 629 21 554 183 252 315 736 525 481 886 848 103 40 593 168 528 968 0
Route 13	0 474 538 532 817 466 930 696 627 994 666 850 777 337 563 424 289 122 306 27 504 875 277 176 638 59 631 166 442 903 599 382 94 987 452 0
Route 14	0 559 236 590 111 222 386 177 673 379 197 776 640 38 258 95 572 951 209 856 809 857 250 206 278 253 780 904 711 127 371 888 574 765 299 433 388 648 0
Route 15	0 602 89 702 221 649 916 722 359 854 26 159 566 501 148 262 132 475 889 86 339 362 400 105 536 682 347 185 246 169 939 53 350 73 342 651 0
Route 16	0 637 309 732 210 203 406 575 437 415 219 58 438 408 88 922 932 143 224 861 63 846 254 749 295 365 77 427 44 896 738 531 617 454 899 0
Route 17	0 665 241 873 82 609 719 561 644 698 317 248 734 152 62 257 786 459 229 412 162 234 76 429 57 340 915 398 811 758 255 905 647 937 462 718 237 913 401 0
Route 18	0 674 394 509 230 580 130 303 106 269 385 689 805 195 507 825 413 260 16 311 93 679 523 357 537 879 669 54 512 436 79 800 541 697 110 446 835 589 0
Route 19	0 676 149 374 403 494 907 20 390 960 78 827 352 67 96 746 652 491 187 245 157 731 335 584 312 348 146 775 519 958 61 807 962 728 478 0
Route 20	0 752 283 240 869 773 837 578 562 771 843 838 204 343 924 547 488 266 920 265 814 188 453 641 936 555 999 1 720 808 8 524 971 623 506 329 461 0
Route 21	0 823 293 667 611 46 492 931 594 85 498 497 714 624 123 966 286 982 514 556 411 4 870 469 375 995 320 569 520 322 852 670 298 993 792 356 318 0
Route 22	0 865 540 911 829 567 713 142 141 314 526 948 341 261 816 300 432 751 114 919 940 296 596 797 628 170 769 974 836 964 783 332 668 973 781 0
Route 23	0 881 102 740 60 694 476 551 855 944 41 180 3 243 595 860 833 138 996 938 186 522 626 564 272 440 828 618 812 378 399 925 256 582 220 535 397 891 333 0
Route 24	0 897 893 74 18 553 933 643 620 17 121 480 787 407 151 158 768 273 493 484 658 29 677 721 992 275 985 605 914 281 313 963 422 449 550 683 636 231 0
Route 25	0 898 455 876 124 918 685 465 451 894 701 910 416 11 700 52 244 496 331 205 655 259 515 952 678 70 909 201 164 499 546 1000 211 435 351 395 704 779 0
Route 26	0 921 387 445 690 686 942 976 486 798 743 639 739 325 430 901 573 591 603 724 174 330 56 778 482 634 84 128 328 529 844 902 675 975 19 560 579 533 633 336 0
Route 27	0 957 345 530 450 988 592 883 233 953 516 726 989 789 72 650 457 369 815 853 147 32 785 681 774 131 392 990 885 5 661 190 355 871 699 13 588 757 0
Route 28	0 959 671 613 473 804 872 194 585 239 441 755 511 28 23 794 428 890 513 867 310 80 99 292 900 542 680 126 878 282 202 35 884 882 0

Instanz R1_200



Knotenanzahl	201
Maximalkapazität	200
Distanz	2869,66
Routenanzahl	18
Route 1	0 4 151 141 43 85 191 129 70 15 188 56 175 9 0
Route 2	0 29 96 65 94 12 82 47 23 178 19 198 114 74 39 0
Route 3	0 46 164 54 125 106 89 110 37 108 0
Route 4	0 60 183 62 194 34 30 128 142 98 20 163 159 100 104 38 32 0
Route 5	0 63 116 5 3 130 11 131 66 50 158 172 0
Route 6	0 87 181 146 169 48 92 185 113 31 69 0
Route 7	0 97 57 10 13 84 90 138 52 45 0
Route 8	0 118 152 88 99 112 121 145 111 35 101 0
Route 9	0 119 105 73 59 196 24 140 0
Route 10	0 143 123 21 26 126 51 156 176 120 0
Route 11	0 147 135 93 199 109 28 182 75 41 91 0
Route 12	0 149 81 134 1 165 103 124 154 0
Route 13	0 155 67 25 61 117 160 86 49 180 195 27 133 64 80 0
Route 14	0 171 16 115 162 170 187 78 139 168 148 0
Route 15	0 174 22 58 127 77 95 177 189 6 102 136 173 83 0
Route 16	0 184 166 33 193 186 144 179 71 190 153 167 72 107 157 0
Route 17	0 192 53 200 40 17 2 76 42 8 122 161 0
Route 18	0 197 14 18 79 44 68 36 137 150 55 7 132 0

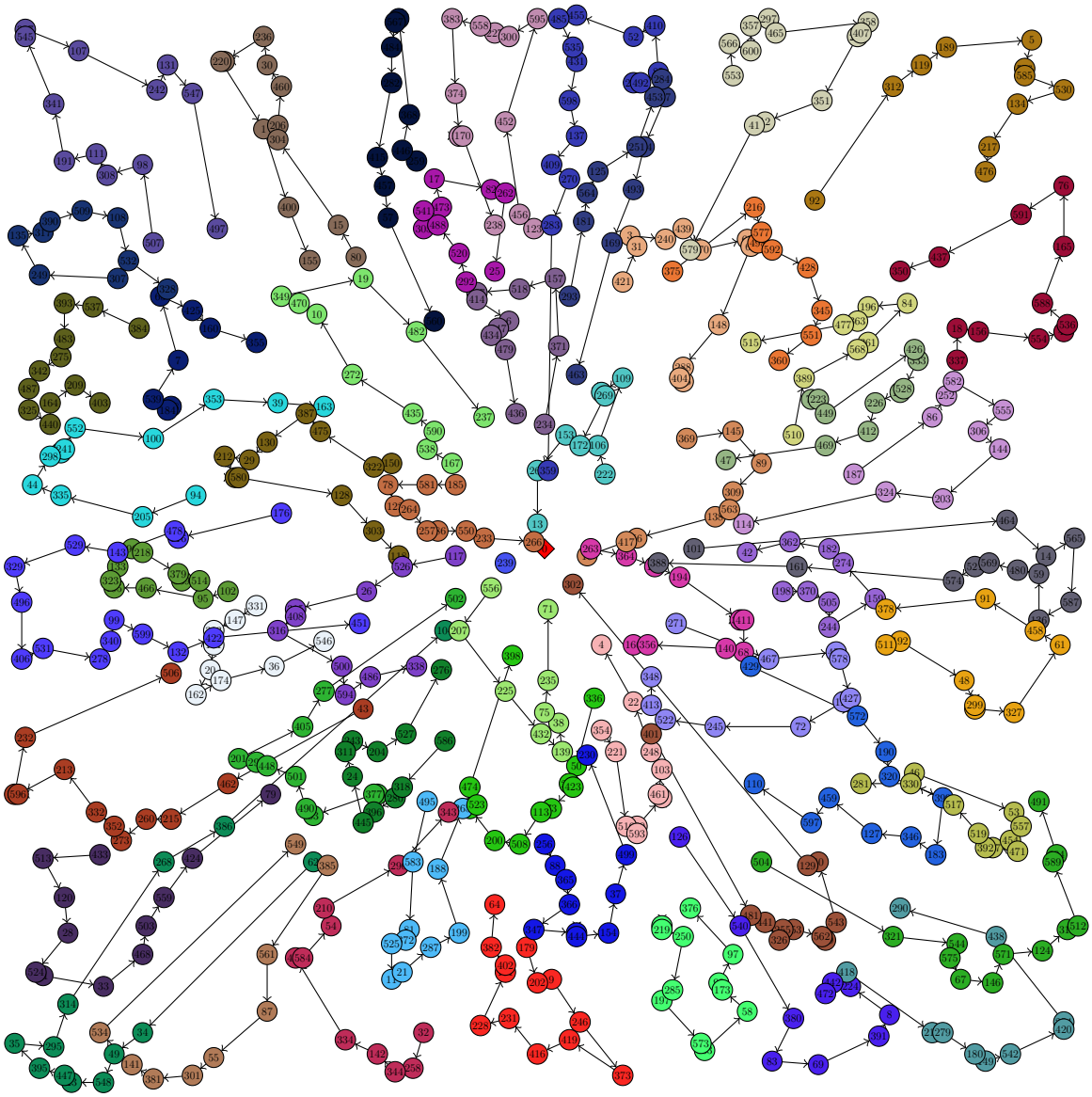
Instanz R1_400



Knotenanzahl	401
Maximalkapazität	200
Distanz	7176,31
Routenanzahl	36
Route 1	0 10 317 24 150 50 48 39 166 274 324 0
Route 2	0 27 160 136 66 231 387 3 255 102 41 119 69 0
Route 3	0 35 121 275 52 211 117 372 268 360 395 0
Route 4	0 44 289 79 5 2 383 59 229 175 279 204 0
Route 5	0 46 212 61 286 295 333 247 321 55 271 113 0
Route 6	0 58 93 205 62 264 252 198 200 83 12 202 14 0
Route 7	0 76 376 258 130 394 291 98 344 250 276 0
Route 8	0 84 101 256 272 6 370 140 308 149 306 30 300 17 94 0
Route 9	0 105 392 1 80 332 18 157 348 248 351 123 358 0
Route 10	0 116 359 177 282 382 187 235 71 251 232 288 185 0
Route 11	0 128 129 366 74 141 56 240 16 380 127 142 0
Route 12	0 145 4 184 192 312 373 207 390 320 29 144 285 20 148 396 0
Route 13	0 155 301 132 139 22 114 237 238 222 265 32 0
Route 14	0 178 154 158 191 266 219 64 253 290 0
Route 15	0 179 60 54 9 57 209 28 227 323 0
Route 16	0 180 182 199 313 63 314 341 170 233 0
Route 17	0 188 43 244 163 302 90 86 172 0
Route 18	0 228 88 206 165 111 115 242 381 340 384 226 0
Route 19	0 249 294 361 377 75 350 293 37 309 367 171 72 0
Route 20	0 260 326 15 257 354 364 197 398 0

Route 21	0 261 322 70 363 126 135 297 152 283 345 68 195 284 397 0
Route 22	0 270 357 19 210 225 315 259 164 109 391 169 0
Route 23	0 287 125 280 78 181 194 89 213 224 220 353 0
Route 24	0 296 31 305 221 25 49 38 96 153 0
Route 25	0 298 281 304 236 216 218 91 138 168 0
Route 26	0 303 193 131 379 36 21 77 65 97 147 325 347 106 0
Route 27	0 318 81 277 328 196 246 99 23 352 208 176 223 159 0
Route 28	0 329 399 174 100 299 356 365 239 371 374 254 112 267 0
Route 29	0 335 319 349 7 85 245 124 311 108 51 0
Route 30	0 336 120 203 217 369 307 167 34 82 327 189 0
Route 31	0 337 107 186 137 334 42 40 11 104 0
Route 32	0 342 346 173 378 67 201 134 230 87 269 161 143 0
Route 33	0 355 241 292 331 343 53 103 183 385 316 263 389 118 0
Route 34	0 368 95 122 133 146 310 215 33 13 8 400 73 0
Route 35	0 386 243 362 45 47 273 190 26 338 214 92 162 0
Route 36	0 388 156 330 234 151 262 339 393 110 278 375 0

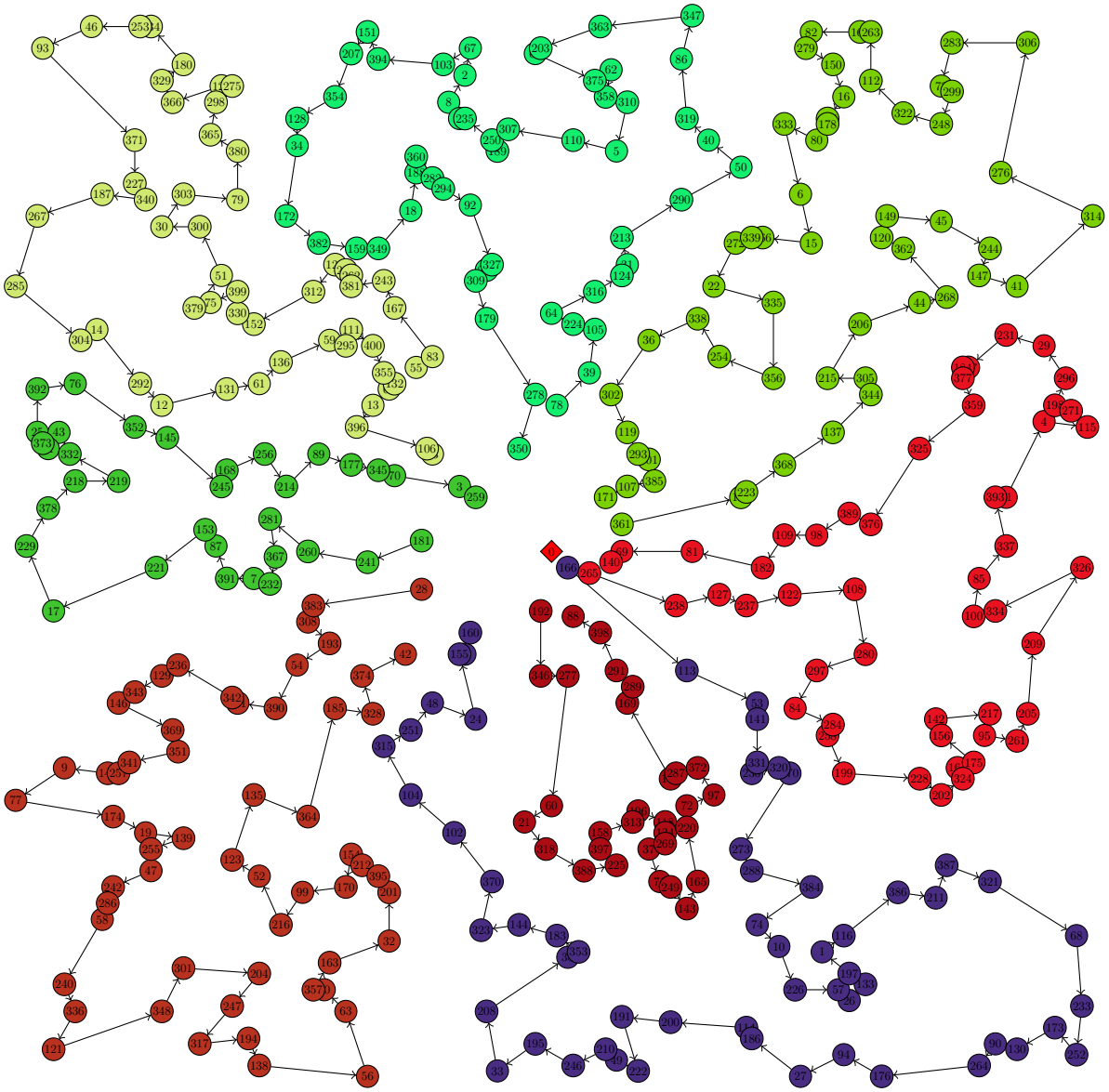
Instanz R1_600



Knotenanzahl	601
Maximalkapazität	200
Distanz	15517,54
Routenanzahl	55
Route 1	0 32 258 344 142 334 494 584 54 210 296 343 0

Route 2	0 43 462 215 260 273 352 332 213 596 115 232 506 0
Route 3	0 46 53 557 454 471 367 392 519 517 330 281 0
Route 4	0 62 34 49 548 23 447 395 35 295 314 268 386 104 0
Route 5	0 74 223 449 426 333 528 158 226 412 469 47 0
Route 6	0 80 15 304 206 460 30 236 70 220 1 400 155 0
Route 7	0 92 312 119 189 5 576 585 530 134 217 476 0
Route 8	0 94 205 335 44 298 152 241 552 100 353 39 163 0
Route 9	0 101 464 14 565 587 136 59 480 569 521 574 161 388 0
Route 10	0 102 95 466 195 323 133 96 218 379 339 514 0
Route 11	0 117 526 26 315 408 316 500 594 486 338 0
Route 12	0 123 456 452 595 300 227 558 383 374 105 170 238 0
Route 13	0 126 540 380 83 69 391 8 224 442 472 0
Route 14	0 150 322 475 387 130 29 212 77 580 128 303 118 0
Route 15	0 167 538 590 435 272 10 470 349 19 482 237 0
Route 16	0 176 116 478 143 529 329 496 406 531 278 340 99 599 132 422 451 0
Route 17	0 177 219 250 285 197 573 533 58 173 90 97 376 0
Route 18	0 178 184 539 7 65 425 160 355 0
Route 19	0 179 202 9 246 373 419 416 231 228 171 402 382 64 0
Route 20	0 185 581 78 122 264 257 186 550 233 266 0
Route 21	0 187 86 252 582 555 306 144 203 324 114 0
Route 22	0 198 370 505 244 159 274 182 362 42 0
Route 23	0 208 377 93 490 501 448 294 201 405 277 502 0
Route 24	0 222 106 109 60 269 172 153 261 13 0
Route 25	0 234 371 157 518 267 414 6 247 434 479 436 0
Route 26	0 239 0
Route 27	0 256 88 365 366 347 443 444 154 37 499 230 0
Route 28	0 259 446 368 291 567 484 282 415 457 57 560 0
Route 29	0 263 364 112 194 310 411 68 140 356 166 0
Route 30	0 271 467 450 578 427 121 72 245 522 413 348 0
Route 31	0 286 492 85 410 52 455 485 535 431 598 137 409 270 283 359 0
Route 32	0 292 520 488 305 541 473 17 82 262 25 0
Route 33	0 293 181 564 125 251 453 284 27 214 493 169 463 0
Route 34	0 307 249 135 317 390 509 108 532 328 0
Route 35	0 331 147 313 151 20 162 174 36 546 0
Route 36	0 336 50 229 423 73 113 508 200 193 523 474 398 0
Route 37	0 337 18 156 554 45 536 588 165 76 591 437 350 0
Route 38	0 354 221 516 593 430 2 461 103 248 22 4 0
Route 39	0 369 145 89 309 563 138 56 417 16 0
Route 40	0 375 216 577 498 592 428 345 551 360 0
Route 41	0 384 537 393 483 275 342 487 325 440 164 209 403 0
Route 42	0 385 561 87 55 301 381 141 534 549 0
Route 43	0 401 481 441 326 255 253 562 489 543 40 129 302 0
Route 44	0 418 211 279 180 149 542 420 254 438 290 0
Route 45	0 421 31 3 240 439 570 63 66 148 288 404 175 0
Route 46	0 429 572 190 320 399 183 346 127 459 597 110 0
Route 47	0 433 513 120 28 524 51 33 468 503 559 424 79 0
Route 48	0 495 583 81 372 525 11 21 287 199 188 265 0
Route 49	0 504 321 544 575 67 146 571 124 319 512 589 394 491 0
Route 50	0 507 98 308 111 191 341 545 397 107 242 131 547 497 0
Route 51	0 510 389 568 361 84 196 363 477 515 0
Route 52	0 511 192 48 299 168 327 61 458 91 378 0
Route 53	0 553 566 600 357 297 465 358 407 289 351 12 41 579 0
Route 54	0 556 207 225 432 139 38 75 235 71 0
Route 55	0 586 318 280 396 445 24 311 243 204 527 276 0

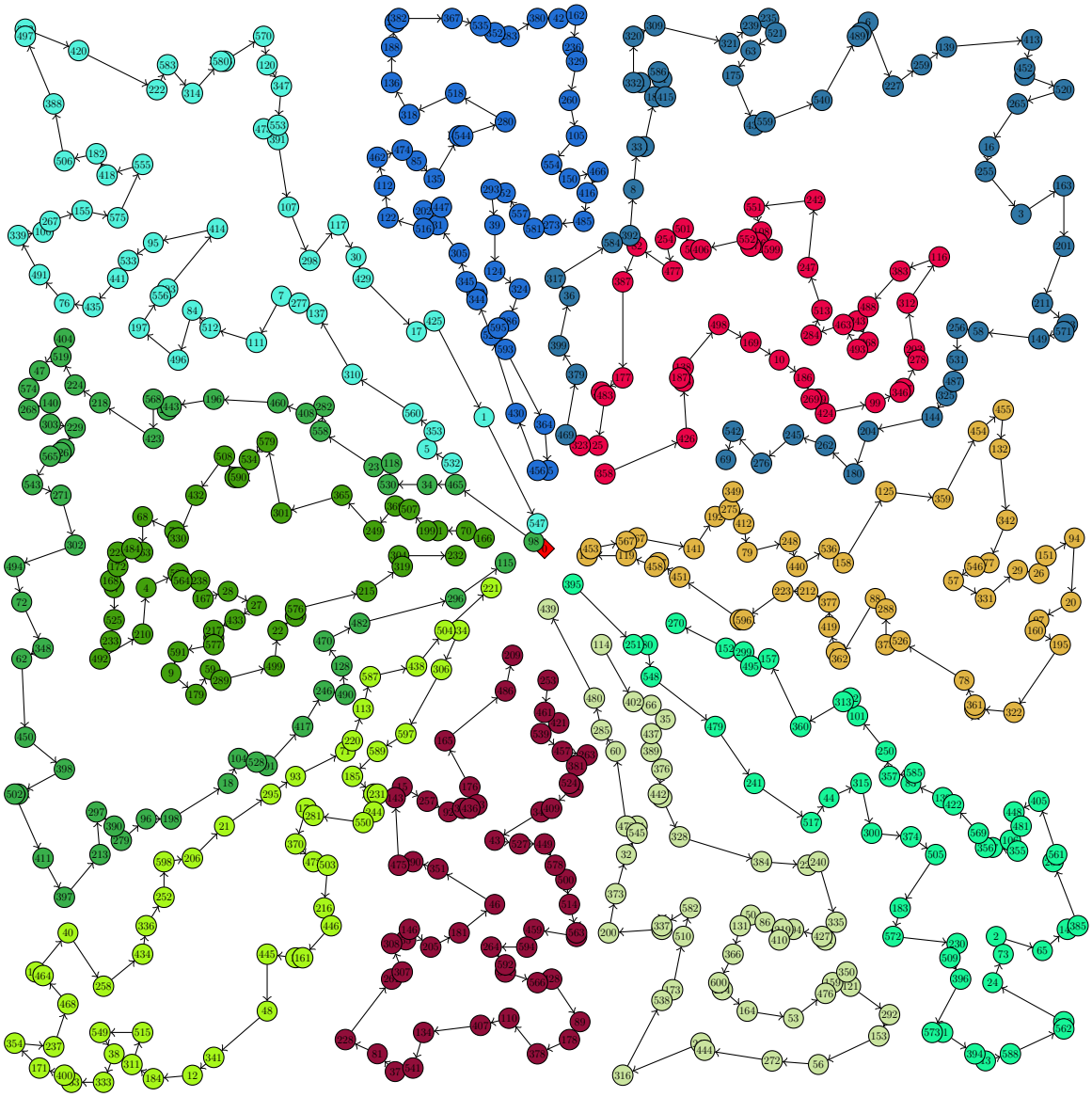
Instanz R2_400



Knotenanzahl	401
Maximalkapazität	1000
Distanz	3312,87
Routenanzahl	8
Route 1	0 28 383 308 193 54 390 311 342 236 129 343 146 369 351 341 257 148 9 77 174 19 139 255 47 242 286 58 240 336 121 348 301 204 247 317 194 138 56 63 20 357 163 32 201 395 212 154 170 99 216 52 123 135 364 185 328 374 42 0
Route 2	0 55 83 167 243 381 262 239 126 312 152 330 399 75 379 51 300 30 303 79 380 365 298 275 125 366 329 180 234 253 46 93 371 227 340 187 267 285 304 14 292 12 131 61 136 59 295 111 400 355 132 65 13 396 106 23 0
Route 3	0 78 39 105 224 64 316 124 31 213 290 50 40 319 86 347 363 203 66 375 62 358 310 5 110 307 189 250 235 96 8 2 67 103 394 151 207 354 128 34 172 382 159 349 18 188 360 282 294 92 327 274 309 179 278 350 0
Route 4	0 166 113 53 141 331 230 320 270 273 288 384 74 10 226 57 26 133 197 1 116 386 211 387 321 68 233 252 173 130 90 264 176 94 27 186 114 200 191 222 49 210 246 195 33 208 38 353 183 144 323 370 102 104 315 251 48 24 155 35 160 0
Route 5	0 181 241 260 281 367 232 7 391 87 153 221 17 229 378 218 219 332 43 91 373 25 392 76 352 145 245 168 256 214 89 177 345 70 3 259 0
Route 6	0 192 346 277 60 21 318 388 225 397 158 313 196 118 134 269 37 71 249 143 165 220 72 97 372 287 190 169 289 291 398 88 0
Route 7	0 265 238 127 237 122 108 280 297 84 284 258 199 228 202 324 164 175 156 142 217 95 261 205 209 326 334 100 85 337 393 11 4 115 271 198 296 29 231 117 184 377 359 325 376 389 98 109 182 81 69 140 0

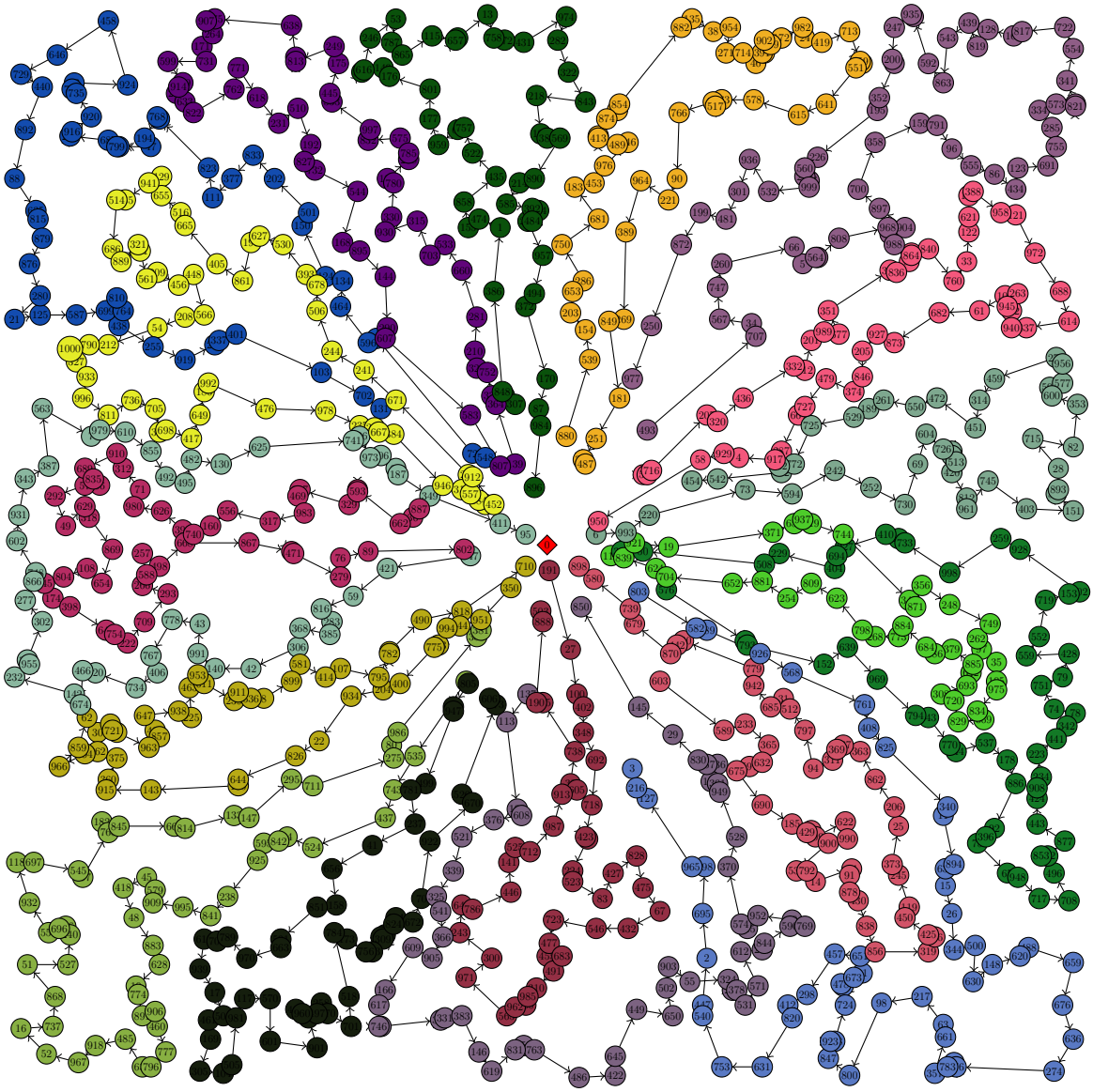
Route 8	0 361 162 223 368 137 344 305 215 206 44 268 362 120 149 45 244 147 41 314 276 306 283 73 299 248 322 112 263 161 82 279 150 16 157 178 80 333 6 15 266 339 272 22 335 356 254 338 36 302 119 293 101 385 107 171 0

Instanz R2_600



Knotenanzahl	601
Maximalkapazität	1000
Distanz	6188,39
Routenanzahl	11
Route 1	0 98 465 34 530 118 23 558 282 408 460 196 443 207 568 423 218 224 519 404 47 574 268 140 303 229 75 326 565 543 271 302 494 72 348 62 450 398 170 502 411 397 213 297 390 279 96 198 18 104 528 291 417 246 490 128 470 482 296 115 0
Route 2	0 114 402 66 35 437 389 376 442 90 328 384 226 240 335 191 427 194 219 410 86 50 131 366 600 274 164 53 476 159 350 121 292 153 56 272 444 214 316 538 173 510 582 337 208 200 373 32 545 471 478 60 285 480 439 0
Route 3	0 166 70 61 199 507 369 249 365 301 579 534 590 123 508 432 330 74 68 363 484 225 172 168 67 525 233 492 210 4 529 564 238 167 28 27 433 217 49 577 591 9 179 59 289 499 22 523 576 215 319 304 232 0
Route 4	0 253 461 421 539 457 263 381 524 511 409 340 43 527 449 578 500 514 563 294 459 594 264 592 371 566 428 89 178 378 110 407 134 541 37 81 228 261 307 308 55 146 205 181 46 351 290 475 143 15 257 92 327 436 343 176 165 486 209 0
Route 5	0 334 306 597 589 185 126 231 244 550 281 174 370 472 503 216 446 161 19 445 48 341 12 184 311 515 549 38 333 133 400 171 354 237 468 464 109 40 258 434 336 252 598 206 21 295 93 71 220 113 587 438 504 221 0
Route 6	0 358 426 103 187 138 498 169 10 186 269 129 424 99 346 287 278 203 312 116 383 488 243 368 493 463 284 513 247 242 551 108 599 386 537 552 406 54 501 254 477 82 387 177 393 483 25 323 0
Route 7	0 395 251 80 548 479 241 517 44 315 300 374 505 183 572 230 509 396 573 51 394 13 588 562 338 24 73 2 65 145 385 234 561 405 448 481 106 355 87 356 569 422 130 83 585 357 250 101 102 313 360 157 495 299 152 270 0
Route 8	0 453 567 467 141 192 275 349 412 79 248 440 536 158 125 359 454 455 132 342 77 546 57 331 29 26 151 94 20 97 160 195 322 127 361 78 526 375 288 88 362 14 419 377 212 223 596 148 451 147 458 119 154 0
Route 9	0 456 430 522 344 142 345 305 31 447 202 516 122 112 462 474 85 135 401 544 280 518 318 136 188 266 382 367 535 352 283 380 42 162 236 329 260 105 554 150 466 416 485 273 581 557 52 293 39 124 324 286 595 593 364 45 0
Route 10	0 469 379 399 36 317 584 392 8 33 11 189 415 41 586 91 332 320 309 321 239 235 521 63 175 431 559 540 489 190 6 227 259 139 413 452 64 520 265 16 255 3 163 201 211 193 571 149 58 256 531 487 325 144 204 180 262 245 276 542 69 0
Route 11	0 532 5 353 560 310 137 277 7 111 512 84 496 197 556 403 414 95 533 441 435 76 491 339 100 267 155 575 555 418 182 506 388 497 372 420 222 583 314 580 156 570 120 347 553 473 391 107 298 117 30 429 17 425 1 547 0

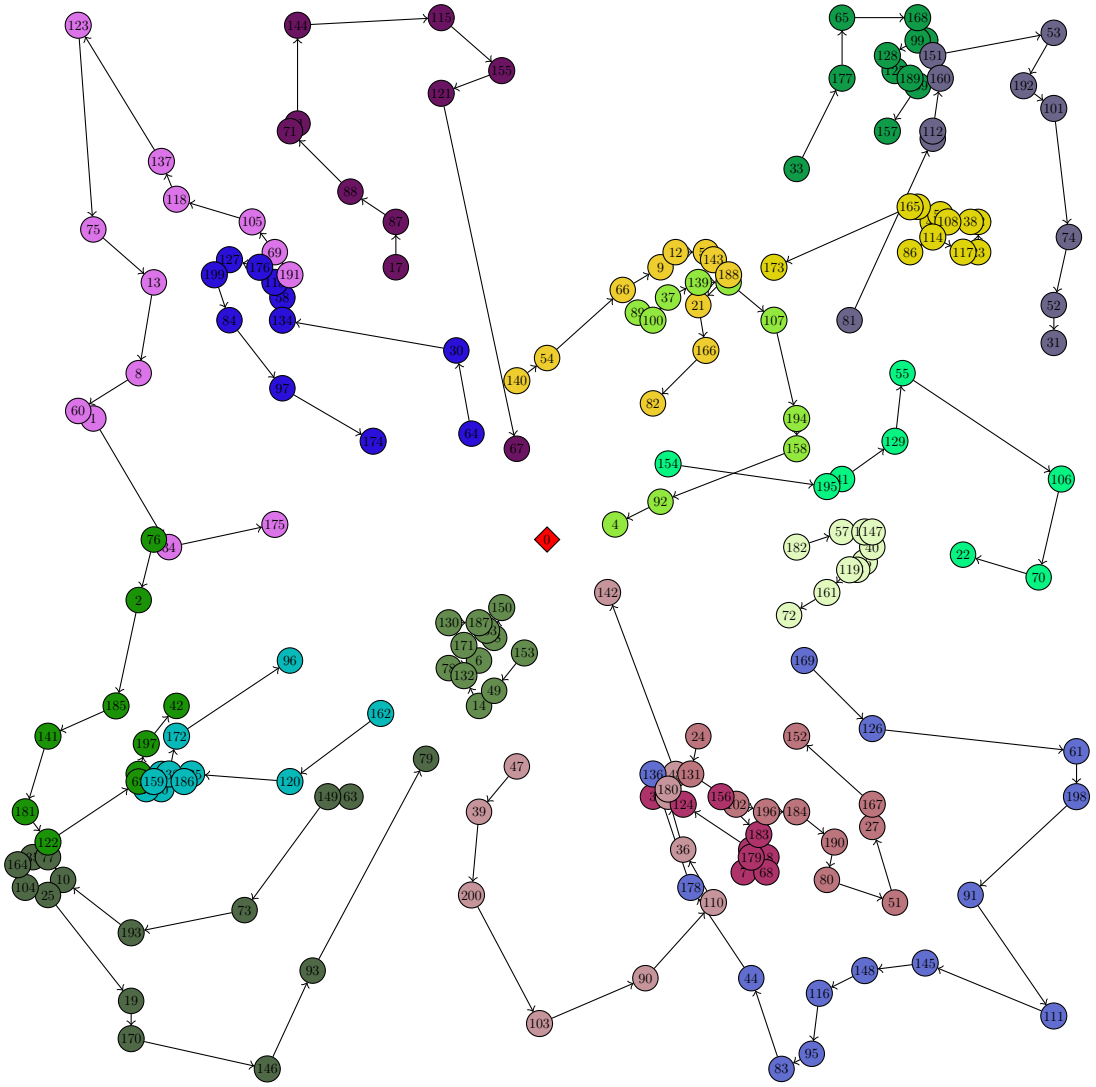
Instanz R2_1000



Knotenanzahl	1001
Maximalkapazität	1000
Distanz	15090,29
Routenanzahl	19
Route 1	0 6 993 220 73 594 242 252 730 69 604 726 101 511 513 420 812 961 745 403 151 584 893 28 715 82 353 600 509 577 956 270 459 314 451 472 550 261 189 529 725 772 852 462 542 454 0
Route 2	0 9 520 716 207 320 436 332 112 201 989 677 351 380 836 99 864 680 840 760 33 122 621 156 388 958 121 972 688 614 837 940 162 945 263 109 61 682 873 927 205 846 374 479 727 664 287 917 4 929 58 950 0
Route 3	0 137 113 193 608 376 521 339 325 541 366 905 609 166 617 746 507 328 331 383 146 619 831 253 763 486 422 645 449 650 502 903 55 324 297 378 531 571 612 256 844 590 769 40 952 515 574 370 528 949 361 157 136 467 830 29 145 850 0
Route 4	0 139 364 338 752 323 210 281 660 533 703 315 930 330 780 126 288 785 575 832 997 333 445 175 249 813 228 638 875 907 264 171 731 599 549 914 291 633 822 762 771 618 231 510 192 827 732 544 168 895 144 290 607 583 807 0
Route 5	0 191 27 100 402 348 692 718 423 299 224 523 83 427 828 475 67 432 546 723 477 455 683 491 310 985 860 962 565 971 300 243 648 786 446 141 525 712 987 913 605 32 738 165 190 888 503 0
Route 6	0 307 303 848 386 1 474 155 858 435 522 757 60 959 177 801 176 149 562 616 246 53 787 865 115 657 110 13 758 572 431 974 282 322 843 218 198 382 569 890 214 585 392 354 484 265 957 494 372 170 87 984 896 0
Route 7	0 350 951 944 994 553 775 400 204 934 22 826 644 180 143 915 360 375 362 394 966 859 62 30 44 721 12 963 857 433 647 938 225 463 953 611 239 911 536 8 899 581 414 107 795 782 77 490 818 710 0

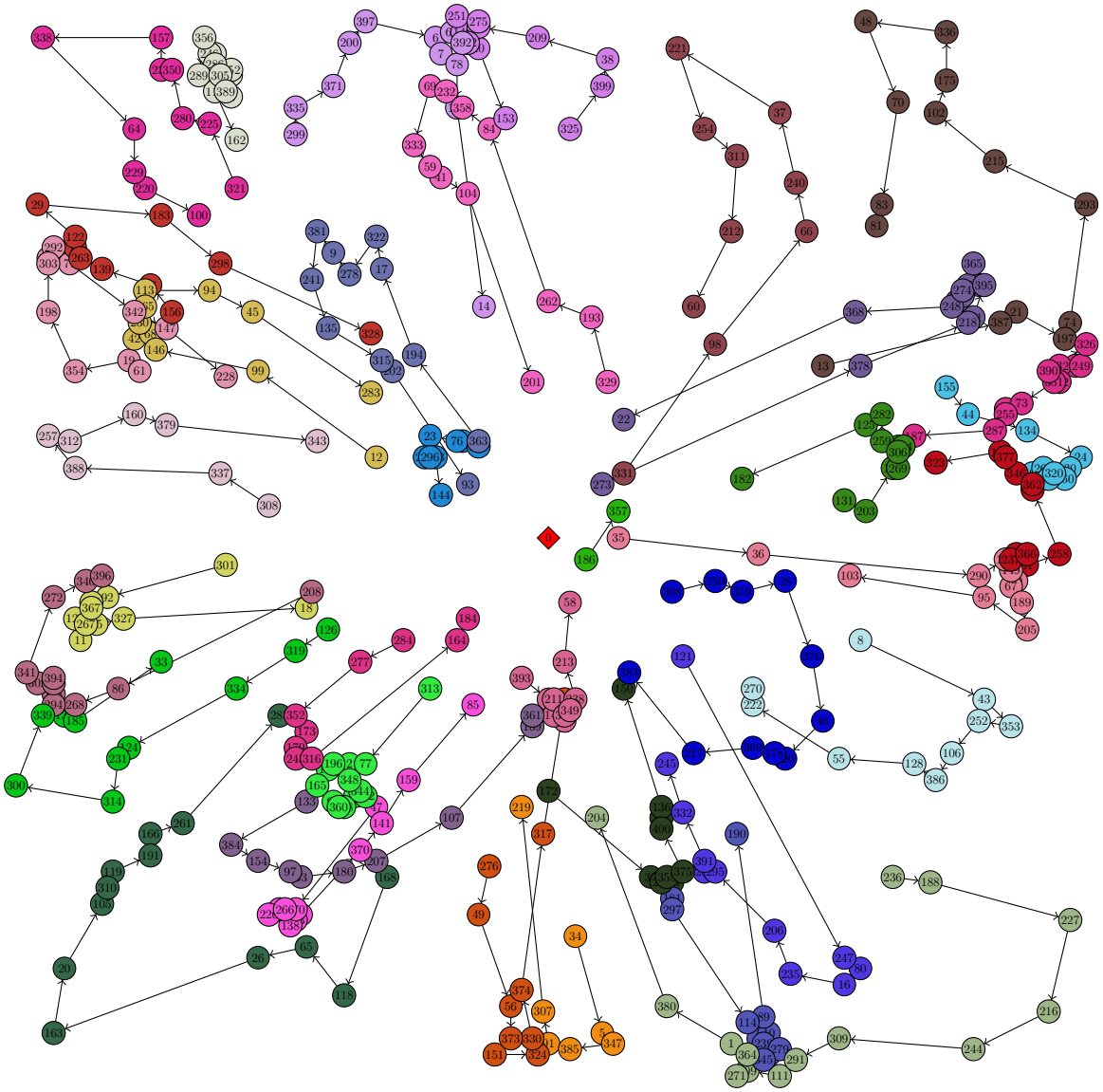
Route 8	0 415 535 743 437 524 84 842 595 925 238 841 995 909 579 45 418 48 883 628 10 774 891 906 460 777 796 634 485 918 967 52 16 737 868 51 527 240 236 696 558 932 118 697 545 407 765 182 845 666 814 133 147 295 711 275 80 986 336 381 0
Route 9	0 452 912 294 671 241 244 506 678 393 530 627 196 861 405 665 516 655 129 941 465 514 686 889 321 161 561 209 456 448 566 208 54 212 790 1000 327 933 996 811 736 705 395 698 417 649 186 992 476 978 235 284 667 384 946 316 557 184 179 0
Route 10	0 493 707 34 567 747 260 66 5 564 335 808 968 483 988 904 897 700 358 159 791 96 555 86 434 123 691 755 285 334 573 821 211 341 554 722 817 164 128 819 439 543 863 592 85 935 247 200 23 352 195 226 560 276 197 999 532 936 301 481 199 872 250 977 0
Route 11	0 547 421 59 816 283 385 368 306 42 140 991 43 778 767 406 734 120 466 674 142 232 955 304 302 277 866 748 602 931 343 387 563 7 979 610 855 492 495 482 130 625 741 426 973 806 20 187 349 411 95 0
Route 12	0 548 72 132 596 464 134 124 150 501 202 833 377 111 823 106 768 194 47 788 799 687 272 916 920 735 728 93 924 458 646 729 440 892 88 635 815 879 876 280 21 125 587 699 810 764 438 255 919 278 337 401 103 702 131 0
Route 13	0 576 167 793 152 639 969 794 643 770 444 537 178 886 92 396 75 65 948 717 708 496 853 742 877 443 424 908 234 223 441 342 78 74 751 79 428 559 552 719 153 102 928 259 998 733 219 410 227 694 404 229 508 50 390 0
Route 14	0 603 589 233 365 632 597 675 690 185 429 68 409 622 990 900 534 792 14 64 91 878 430 838 856 319 416 425 450 119 245 373 25 206 862 363 367 369 311 94 797 512 31 685 942 779 230 442 870 679 739 580 898 0
Route 15	0 803 582 289 926 568 761 408 825 340 11 894 637 15 26 344 500 630 148 620 488 659 676 636 274 776 538 783 357 661 63 217 98 800 847 923 188 724 215 478 673 81 651 457 298 412 820 631 753 540 447 2 695 598 965 127 216 3 0
Route 16	0 805 943 947 499 781 237 41 656 158 851 591 663 970 789 706 613 939 17 504 461 169 305 104 505 981 117 570 601 901 326 960 480 497 258 470 701 518 784 173 756 473 309 267 138 824 114 672 70 922 670 526 273 606 0
Route 17	0 880 539 154 203 653 286 750 681 183 453 976 346 489 413 296 874 355 854 882 135 38 954 271 714 46 391 213 902 759 172 982 24 419 713 359 18 551 641 615 578 163 517 468 766 90 221 964 389 269 849 181 251 313 487 0
Route 18	0 887 36 662 39 593 329 397 469 983 317 556 160 740 399 626 980 71 312 910 689 835 56 57 292 49 629 318 869 654 108 804 345 174 398 640 754 222 709 293 266 588 257 498 668 867 37 471 279 76 89 802 0
Route 19	0 921 19 371 658 937 519 744 347 871 356 248 749 262 97 35 105 975 834 669 829 720 308 693 642 885 379 586 684 884 773 268 798 623 809 254 881 652 704 624 839 116 0

Instanz RC1_200



Knotenanzahl	201
Maximalkapazität	200
Distanz	2800,78
Routenanzahl	18
Route 1	0 17 87 88 71 11 144 115 155 121 67 0
Route 2	0 24 131 102 196 184 190 80 51 27 167 152 0
Route 3	0 33 177 65 168 94 99 128 125 189 109 157 0
Route 4	0 47 39 200 103 90 110 36 43 180 48 142 0
Route 5	0 63 149 73 193 10 77 35 164 104 25 19 170 146 93 79 0
Route 6	0 64 30 134 58 113 176 127 199 84 97 174 0
Route 7	0 76 2 185 141 181 122 62 5 197 42 0
Route 8	0 81 50 112 160 151 53 192 101 74 52 31 0
Route 9	0 86 114 117 23 32 38 108 56 45 85 165 173 0
Route 10	0 89 100 37 139 29 107 194 158 92 4 0
Route 11	0 140 54 66 9 12 59 143 188 21 166 82 0
Route 12	0 153 49 14 132 78 6 171 130 187 163 28 150 0
Route 13	0 154 195 41 129 55 106 70 22 0
Route 14	0 156 183 18 68 7 179 15 124 3 0
Route 15	0 162 120 135 186 20 26 159 46 138 172 96 0
Route 16	0 169 126 61 198 91 111 145 148 116 95 83 44 178 136 0
Route 17	0 182 57 133 147 40 16 98 119 161 72 0
Route 18	0 191 69 105 118 137 123 75 13 8 60 1 34 175 0

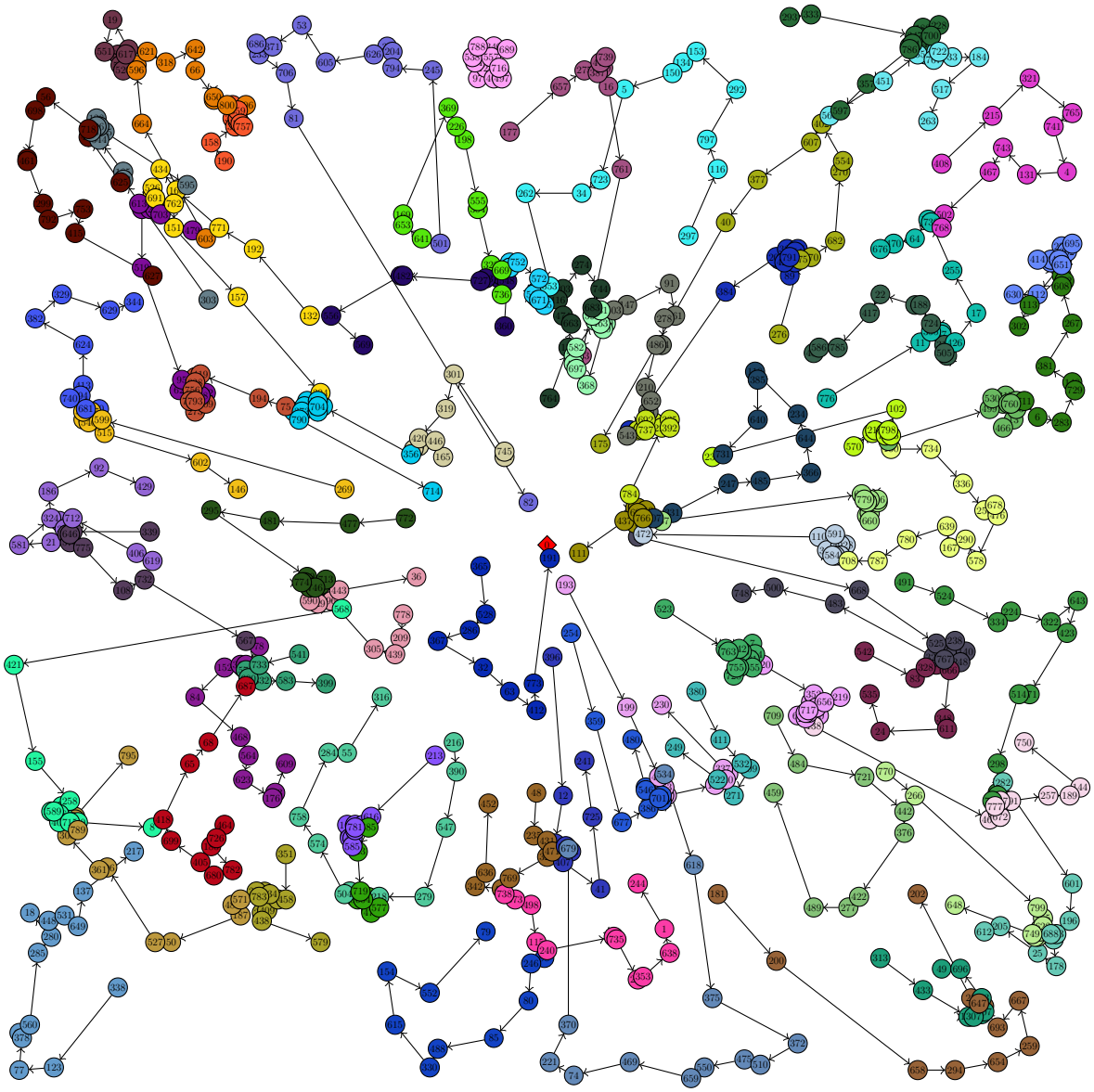
Instanz RC1_400



Knotenanzahl	401
Maximalkapazität	200
Distanz	7267,21
Routenanzahl	37
Route 1	0 8 43 353 252 106 386 128 55 222 270 0
Route 2	0 12 99 146 68 42 230 265 113 94 45 283 0
Route 3	0 13 387 21 197 74 293 215 102 175 336 48 70 83 81 0
Route 4	0 34 5 347 385 91 307 219 0
Route 5	0 35 36 290 158 149 50 67 189 205 95 103 0
Route 6	0 47 170 266 224 226 138 123 52 370 141 159 85 0
Route 7	0 61 19 354 198 303 27 292 51 71 342 147 228 0
Route 8	0 87 389 82 152 305 286 57 246 356 289 116 162 0
Route 9	0 121 247 80 16 235 206 295 288 391 332 245 0
Route 10	0 126 319 334 124 231 314 300 339 137 185 33 0
Route 11	0 131 203 142 269 306 304 63 259 282 125 182 0
Route 12	0 133 384 154 97 53 180 207 107 169 361 0
Route 13	0 155 44 134 24 30 260 320 281 264 199 0
Route 14	0 156 96 139 263 79 122 29 183 298 328 0
Route 15	0 168 118 65 26 163 20 105 310 119 191 166 261 285 0
Route 16	0 172 39 210 355 177 375 400 117 136 150 0
Route 17	0 181 297 114 4 239 110 345 195 279 10 89 190 0
Route 18	0 186 357 0
Route 19	0 208 86 268 294 174 382 394 302 341 272 340 396 0
Route 20	0 214 32 76 23 256 296 143 144 0
Route 21	0 236 188 227 216 244 309 291 111 109 271 364 1 380 204 0

Route 22	0 237 101 366 31 258 233 362 346 377 145 323 0
Route 23	0 273 378 218 171 395 54 365 88 274 223 176 248 368 22 0
Route 24	0 276 49 56 373 151 324 330 374 317 161 0
Route 25	0 284 277 352 173 179 243 316 164 184 0
Route 26	0 299 335 371 200 397 6 62 392 7 78 167 14 0
Route 27	0 301 92 115 367 129 11 267 15 327 18 0
Route 28	0 308 337 388 257 312 160 379 343 0
Route 29	0 313 77 348 344 242 3 108 360 90 165 127 196 2 0
Route 30	0 321 225 280 350 234 157 338 64 229 220 100 0
Route 31	0 325 399 38 209 253 275 251 148 130 372 40 153 0
Route 32	0 329 193 262 84 358 232 69 333 59 41 104 201 0
Route 33	0 331 98 66 240 37 221 254 311 212 60 0
Route 34	0 363 194 17 322 278 9 381 241 135 315 202 93 0
Route 35	0 390 132 326 249 112 192 351 73 25 255 287 187 0
Route 36	0 393 75 211 140 72 318 349 238 213 58 0
Route 37	0 398 250 359 28 376 46 120 178 369 217 383 0

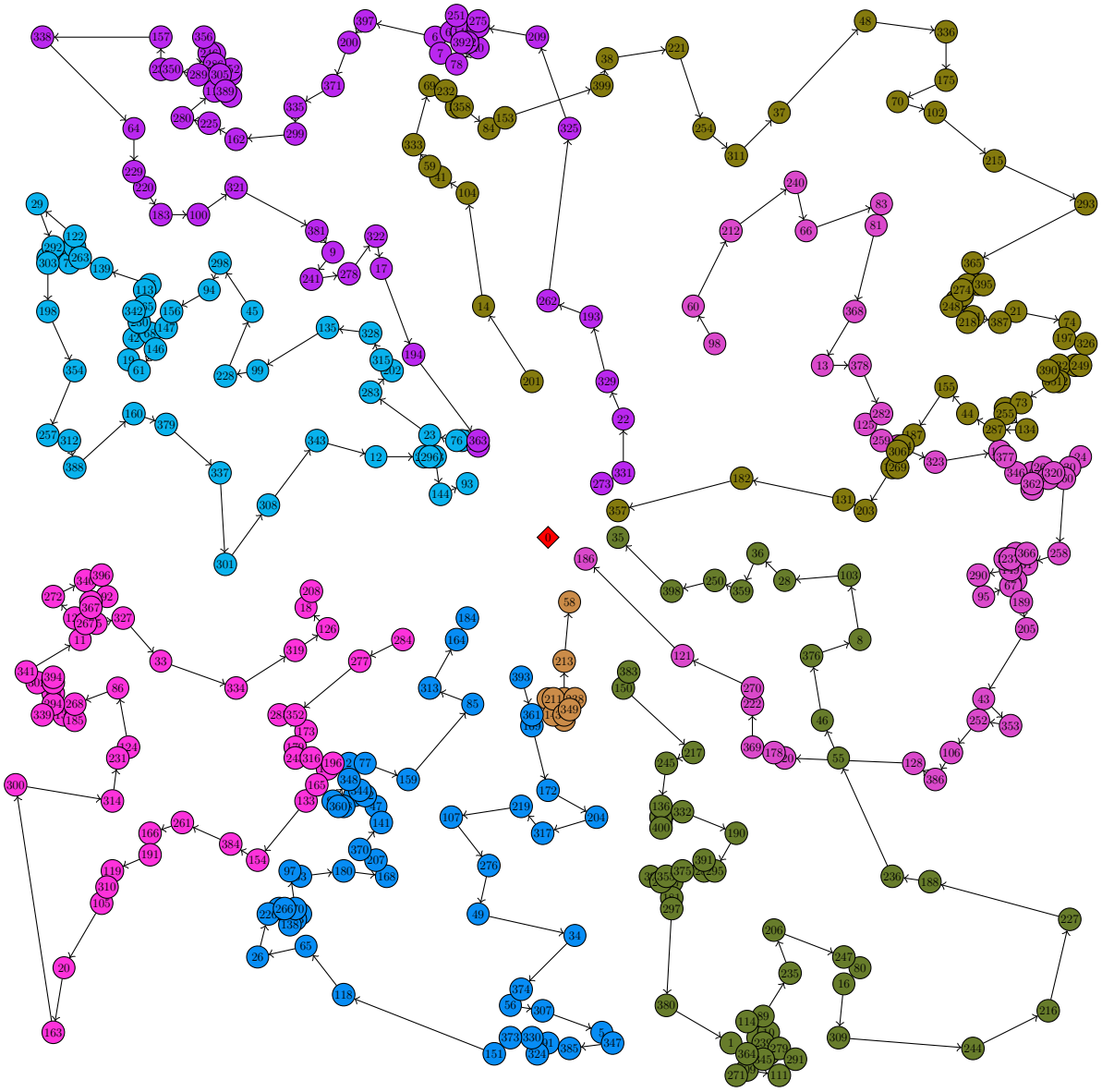
Instanz RC1_800



Knotenanzahl	801
Maximalkapazität	200
Distanz	26655,82
Routenanzahl	73

Route 1	0 3 52 690 236 349 766 73 437 111 0
Route 2	0 48 235 431 471 327 769 427 342 636 452 0
Route 3	0 71 668 525 121 238 140 248 767 326 483 500 748 0
Route 4	0 75 194 419 69 628 756 759 793 273 309 0
Route 5	0 78 310 306 152 84 468 564 623 176 143 609 0
Route 6	0 89 103 454 345 325 791 424 39 201 384 145 0
Route 7	0 90 681 740 494 413 624 382 329 629 344 0
Route 8	0 98 246 80 85 488 330 615 154 552 79 0
Route 9	0 120 352 106 512 656 219 70 717 281 430 222 67 0
Route 10	0 132 192 771 168 434 526 493 691 762 151 157 394 0
Route 11	0 138 462 672 465 777 747 170 291 257 189 144 750 0
Route 12	0 160 734 336 250 678 476 578 290 167 639 780 787 708 0
Route 13	0 177 657 275 473 739 260 387 16 761 133 0
Route 14	0 181 200 658 294 654 259 667 693 20 647 28 202 0
Route 15	0 190 158 383 350 460 559 539 533 592 757 0
Route 16	0 193 199 432 163 42 2 129 440 60 337 230 0
Route 17	0 213 616 393 781 620 449 585 388 51 166 72 183 0
Route 18	0 216 390 547 279 218 208 300 504 574 758 284 55 316 0
Route 19	0 237 505 312 724 188 43 22 417 785 586 456 0
Route 20	0 254 359 677 386 272 701 395 546 398 480 0
Route 21	0 269 599 537 558 58 540 515 602 146 0
Route 22	0 276 675 670 682 270 554 402 607 377 40 175 0
Route 23	0 282 601 196 593 688 136 178 25 205 612 0
Route 24	0 293 333 347 182 228 700 164 684 289 786 357 597 0
Route 25	0 297 116 797 292 153 134 150 5 723 34 262 553 0
Route 26	0 303 253 185 544 252 705 606 128 545 265 595 0
Route 27	0 313 433 174 47 307 100 107 173 696 49 0
Route 28	0 315 499 530 317 251 561 760 588 453 13 466 0
Route 29	0 338 123 77 378 61 560 285 280 18 448 531 649 137 217 0
Route 30	0 339 565 646 511 576 775 108 732 567 0
Route 31	0 346 584 573 496 428 492 591 110 472 0
Route 32	0 351 458 435 634 38 225 783 159 409 243 438 579 0
Route 33	0 356 93 661 704 521 114 358 373 37 790 714 0
Route 34	0 360 548 320 314 563 727 482 45 556 569 0
Route 35	0 365 528 286 367 32 63 412 773 191 0
Route 36	0 380 411 532 509 271 522 249 0
Route 37	0 396 12 341 141 674 122 407 41 725 241 0
Route 38	0 400 497 86 716 557 689 340 788 538 404 124 97 0
Route 39	0 408 215 321 765 741 4 131 743 467 502 768 0
Route 40	0 414 374 397 232 695 362 101 447 59 651 112 630 0
Route 41	0 457 662 779 637 635 516 410 287 264 660 0
Route 42	0 464 726 180 782 680 405 699 418 65 68 687 0
Route 43	0 479 703 256 197 613 179 519 95 62 518 212 0
Route 44	0 491 524 334 224 322 643 423 171 514 298 127 633 754 0
Route 45	0 501 245 794 204 31 626 605 53 371 686 233 706 81 82 0
Route 46	0 503 147 91 161 278 364 486 210 652 391 543 0
Route 47	0 507 331 247 485 366 644 234 118 385 640 731 0
Route 48	0 523 268 763 742 610 7 354 401 655 710 755 728 0
Route 49	0 534 618 375 372 510 475 550 659 469 74 221 370 679 0
Route 50	0 541 733 211 223 575 9 600 30 332 583 399 0
Route 51	0 542 83 328 27 666 348 611 24 535 0
Route 52	0 562 745 301 319 420 99 76 446 165 0
Route 53	0 566 451 665 109 139 707 694 722 33 184 517 263 0
Route 54	0 568 421 155 258 104 220 296 589 231 403 715 490 580 8 0
Route 55	0 570 135 214 702 751 796 798 102 239 0
Route 56	0 571 487 187 50 527 206 361 308 789 119 117 795 0
Route 57	0 603 664 596 261 621 318 642 66 650 425 800 506 0
Route 58	0 617 88 19 551 87 23 508 520 0
Route 59	0 619 406 712 207 125 363 21 581 324 186 92 429 0
Route 60	0 625 445 718 56 698 44 461 299 792 753 415 627 0
Route 61	0 632 54 752 572 94 441 513 105 142 671 645 0
Route 62	0 641 653 169 369 226 198 555 304 323 669 736 0
Route 63	0 685 343 719 389 604 35 311 478 577 0
Route 64	0 697 495 379 582 10 598 631 29 149 463 450 368 0
Route 65	0 709 484 721 442 376 422 277 489 459 0
Route 66	0 711 6 283 729 148 381 267 608 172 46 113 302 0
Route 67	0 738 673 498 115 240 444 735 242 353 638 1 244 0
Route 68	0 764 156 663 474 416 203 274 744 683 0
Route 69	0 770 266 799 455 622 195 594 536 587 749 648 0
Route 70	0 772 477 481 295 549 774 614 720 746 288 713 0
Route 71	0 776 11 355 15 57 426 17 255 730 126 64 470 676 0
Route 72	0 778 209 439 305 96 529 14 590 227 26 443 36 0
Route 73	0 784 229 392 335 130 692 737 436 162 0

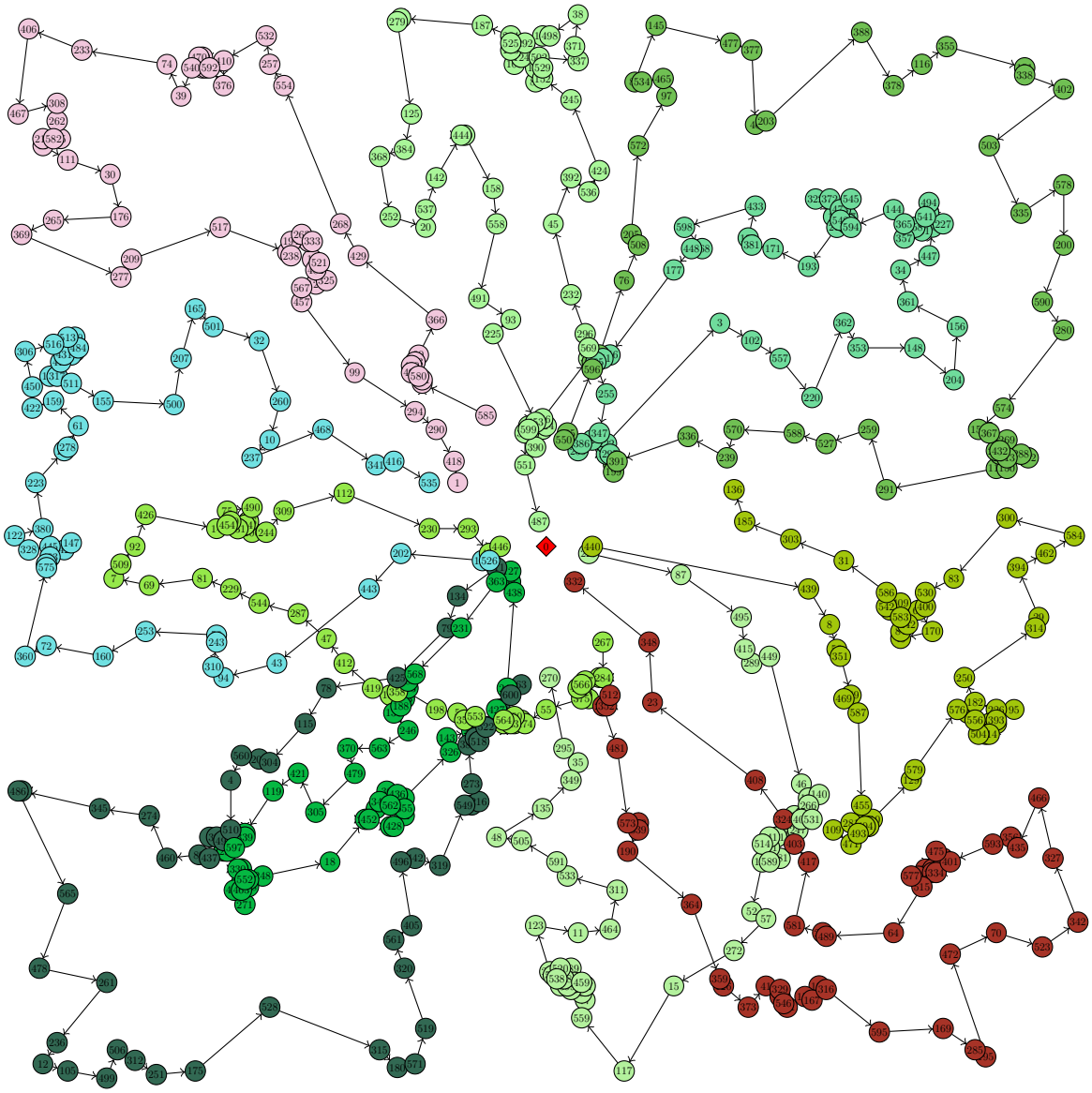
Instanz RC2_400



Knotenanzahl	401
Maximalkapazität	1000
Distanz	3090,19
Routenanzahl	8
Route 1	0 32 76 23 283 202 315 328 135 99 228 45 298 94 156 147 68 146 61 19 42 230 342 265 113 96 139 263 71 51 79 122 29 292 27 303 198 354 257 312 388 160 379 337 301 308 343 12 256 296 143 144 93 0
Route 2	0 75 211 140 72 318 349 238 161 213 58 0
Route 3	0 98 60 212 240 66 83 81 368 13 378 282 125 259 63 323 145 377 346 233 362 199 264 281 320 24 30 260 258 31 366 101 237 158 149 290 95 67 50 189 205 43 353 252 106 386 128 120 178 369 222 270 121 186 0
Route 4	0 201 14 104 41 59 333 69 232 167 358 84 153 399 38 221 254 311 37 48 336 175 70 102 215 293 365 395 54 88 274 223 176 248 218 171 387 21 74 197 326 249 112 132 390 192 351 73 25 255 134 287 44 155 187 304 306 269 142 203 131 182 357 0
Route 5	0 273 331 22 329 193 262 325 209 253 275 251 62 148 130 392 372 40 78 7 6 397 200 371 335 299 162 225 280 116 389 87 82 152 305 286 57 246 356 289 350 234 157 338 64 229 220 183 100 321 381 9 241 278 322 17 194 363 214 0
Route 6	0 284 277 352 285 173 179 243 316 196 127 165 133 154 384 261 166 191 119 310 105 20 163 300 314 231 124 86 268 185 137 339 294 174 382 394 302 341 11 267 129 272 340 396 92 115 367 15 327 33 334 319 126 18 208 0
Route 7	0 383 150 217 245 136 117 400 332 190 295 288 391 375 177 355 39 210 181 297 380 1 364 271 109 111 291 279 195 345 110 239 10 4 114 89 235 206 247 80 16 309 244 216 227 188 236 55 46 376 8 103 28 36 359 250 398 35 0

Route 8	0 393 361 169 172 204 317 219 107 276 49 34 374 56 307 5 347 385 91 324 330 373 151 118 65 26 226 224 266 138 123 52 170 97 53 180 168 207 370 141 47 242 344 3 108 360 90 348 2 77 159 85 313 164 184 0

Instanz RC2_600



Knotenanzahl	601
Maximalkapazität	1000
Distanz	5715,92
Routenanzahl	11
Route 1	0 127 363 231 568 382 173 133 188 138 246 563 370 479 305 421 119 146 339 597 298 89 330 184 552 189 413 463 271 318 397 248 18 283 452 73 343 36 436 562 67 428 22 555 21 326 143 276 427 241 438 44 0
Route 2	0 149 297 222 3 102 557 220 362 353 148 204 156 361 34 447 68 227 494 51 541 131 86 258 357 365 144 594 461 91 307 473 545 372 322 16 548 217 193 171 381 196 433 598 168 448 177 256 451 485 255 347 224 386 235 0
Route 3	0 211 134 79 425 78 115 304 208 560 4 510 497 302 9 437 80 460 274 345 101 486 565 478 261 236 12 105 499 506 312 251 175 528 315 180 571 519 320 561 405 496 442 319 549 98 216 273 383 476 518 522 162 600 63 0
Route 4	0 228 87 495 415 289 449 46 140 132 266 531 206 407 201 247 120 161 441 514 191 181 589 53 137 52 57 272 15 117 559 77 164 197 459 398 215 399 88 458 389 520 538 507 213 123 11 464 311 533 591 505 48 135 349 35 295 270 0
Route 5	0 267 284 62 547 566 488 28 2 375 55 474 6 480 423 543 379 564 553 354 5 198 66 358 104 419 412 47 287 544 229 81 69 7 509 92 426 14 454 340 75 490 186 234 331 100 244 309 112 230 293 434 26 446 0

Route 6	0 390 37 214 346 385 569 296 232 45 392 536 424 245 152 85 107 529 502 337 371 38 498 114 292 124 108 183 525 264 187 50 279 125 384 368 252 20 537 142 444 430 158 558 491 93 225 453 96 599 174 344 551 487 0
Route 7	0 440 439 8 59 351 249 469 587 455 281 109 471 493 404 301 24 110 129 579 576 141 556 504 396 414 323 153 393 95 226 182 250 314 29 394 462 584 300 83 530 400 60 170 49 84 242 583 409 240 17 542 586 31 303 185 136 0
Route 8	0 512 352 275 481 573 179 539 190 364 359 126 373 41 329 172 546 321 154 167 106 316 595 169 285 195 472 70 523 342 327 466 435 356 593 401 139 475 54 334 254 27 13 71 577 515 64 489 286 581 417 403 324 408 23 348 332 0
Route 9	0 526 178 202 443 43 94 310 243 58 253 160 72 360 575 524 483 42 147 445 328 122 380 223 166 278 61 159 422 450 306 516 513 350 484 192 387 431 411 113 317 511 155 500 207 165 501 32 260 10 237 468 341 416 535 0
Route 10	0 550 395 596 456 299 76 508 205 572 97 465 534 82 145 477 377 40 203 388 378 116 355 151 338 402 503 335 578 200 590 280 574 157 367 269 432 282 219 313 288 212 150 118 291 259 527 588 570 239 336 391 103 199 0
Route 11	0 585 33 221 580 492 420 163 90 366 429 268 554 257 532 410 376 592 374 218 130 470 540 39 74 233 406 467 308 262 25 582 210 65 111 30 176 265 369 277 209 517 56 238 194 263 128 333 19 521 482 325 121 567 457 99 294 290 418 1 0

B Tabellen

B.1 Pearson-Korrelationen zu den Anteilen der GPU-Zeit

In diesem Abschnitt finden sich die Tabellen der Korrelationen, die in Abschnitt 4.1.1.1 vorgestellt werden.

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,950**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,950**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.1: Pearson-Korrelation zwischen der Kernelzeit zur Move-Bewertung und der Knotenanzahl (Swap - 128 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,867**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,867**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.2: Pearson-Korrelation zwischen der Kernelzeit zur Bestimmung des besten Moves und der Knotenanzahl (Swap - 128 Threads)

		Knoten	Transferzeit
Knoten	Pearson-Korrelation	1	0,420**
	Sig. (2-seitig)		0,000
	N	900	900
Transferzeit	Pearson-Korrelation	0,420**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.3: Pearson-Korrelation zwischen der Transferzeit und der Knotenanzahl (Swap - 128 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,955**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,955**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.4: Pearson-Korrelation zwischen der Kernelzeit zur Move-Bewertung und der Knotenanzahl (Or-Opt - 448 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,760**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,760**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.5: Pearson-Korrelation zwischen der Kernelzeit zur Bestimmung des besten Moves und der Knotenanzahl (Or-Opt - 448 Threads)

		Knoten	Transferzeit
Knoten	Pearson-Korrelation	1	0,258**
	Sig. (2-seitig)		0,000
	N	900	900
Transferzeit	Pearson-Korrelation	0,258**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.6: Pearson-Korrelation zwischen der Transferzeit und der Knotenanzahl (Or-Opt - 448 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,955**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,955**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.7: Pearson-Korrelation zwischen der Kernelzeit zur Move-Bewertung und der Knotenanzahl (2-Opt* - 416 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,844**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,844**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.8: Pearson-Korrelation zwischen der Kernelzeit zur Bestimmung des besten Moves und der Knotenanzahl (2-Opt* - 416 Threads)

		Knoten	Transferzeit
Knoten	Pearson-Korrelation	1	0,614**
	Sig. (2-seitig)		0,000
	N	900	900
Transferzeit	Pearson-Korrelation	0,614**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.9: Pearson-Korrelation zwischen der Transferzeit und der Knotenanzahl (2-Opt* - 416 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,950**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,950**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.10: Pearson-Korrelation zwischen der Kernelzeit zur Move-Bewertung und der Knotenanzahl (2-Opt - 256 Threads)

		Knoten	Kernelzeit
Knoten	Pearson-Korrelation	1	0,754**
	Sig. (2-seitig)		0,000
	N	900	900
Kernelzeit	Pearson-Korrelation	0,754**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.11: Pearson-Korrelation zwischen der Kernelzeit zur Bestimmung des besten Moves und der Knotenanzahl (2-Opt - 256 Threads)

		Knoten	Transferzeit
Knoten	Pearson-Korrelation	1	0,318**
	Sig. (2-seitig)		0,000
	N	900	900
Transferzeit	Pearson-Korrelation	0,318**	1
	Sig. (2-seitig)	0,000	
	N	900	900

** Korrelation ist signifikant auf dem Niveau 0.01 (2-seitig)

Tabelle B.12: Pearson-Korrelation zwischen der Transferzeit und der Knotenanzahl (2-Opt - 256 Threads)

B.2 Kennzahlen zum Vergleich der sequentiellen und parallelen Verfahren

Die Kennzahlen, die zum Vergleich der sequentiellen zur parallelen Version in Abschnitt 4.1.3 betrachtet werden, finden sich im Folgenden.

Tabelle B.13: Kennzahlen zum Vergleich paralleler und sequentieller Verfahren

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
1-1-Cross-Exchange	101	320	0,00039574	0,00048427	1,22370748	0,20261888	0,00633184	0,20213461	0,00584757	0,00239005	0,07648172
1-1-Cross-Exchange	256	128	0,00078452	0,00281631	3,58985112	0,40167424	0,01255232	0,39885793	0,00973601	0,00701143	0,22436569
1-1-Cross-Exchange	301	416	0,00093798	0,00419837	4,47596964	0,48024576	0,01500768	0,47604739	0,01080931	0,00874213	0,2797481
1-1-Cross-Exchange	324	160	0,00095888	0,00433672	4,52269314	0,49094656	0,01534208	0,48660984	0,01100536	0,00883339	0,28266832
1-1-Cross-Exchange	400	128	0,00105646	0,00663265	6,27818375	0,54090752	0,01690336	0,53427487	0,01027071	0,01226208	0,39238648
1-1-Cross-Exchange	484	352	0,00106377	0,00935985	8,79875349	0,54465024	0,01702032	0,53529039	0,00766047	0,01718507	0,54992209
1-1-Cross-Exchange	601	320	0,00140953	0,01418963	10,06692302	0,72167936	0,02255248	0,70748973	0,00836285	0,01966196	0,62918269
1-1-Cross-Exchange	801	352	0,0019636	0,02381404	12,12774496	1,0053632	0,0314176	0,98154916	0,00760356	0,023687	0,75798406
1-1-Cross-Exchange	1001	96	0,00268695	0,04100552	15,26099109	1,3757184	0,0429912	1,33471288	0,00198568	0,02980662	0,95381194
1-2-Cross-Exchange	101	160	0,00039879	0,00039183	0,98254721	0,20418048	0,00638064	0,20378865	0,00598881	0,00191904	0,0614092
1-2-Cross-Exchange	256	96	0,000866	0,00230183	2,65800231	0,443392	0,013856	0,44109017	0,01155417	0,00519141	0,16612514
1-2-Cross-Exchange	301	416	0,00096294	0,00317238	3,29447318	0,49302528	0,01540704	0,4898529	0,01223466	0,00643452	0,20590457
1-2-Cross-Exchange	324	160	0,00097956	0,00371505	3,79257013	0,50153472	0,01567296	0,49781967	0,01195791	0,00740736	0,23703563
1-2-Cross-Exchange	400	128	0,00104564	0,00566256	5,41540109	0,53536768	0,01673024	0,52970512	0,01106768	0,01057696	0,33846257
1-2-Cross-Exchange	484	352	0,00108183	0,00774545	7,15958145	0,55389696	0,01730928	0,54615151	0,00956383	0,01398356	0,44747384
1-2-Cross-Exchange	601	320	0,00140769	0,01183486	8,40729138	0,72073728	0,02252304	0,70890242	0,01068818	0,01642049	0,52545571
1-2-Cross-Exchange	801	352	0,00198462	0,02037716	10,26753736	1,01612544	0,03175392	0,99574828	0,01137676	0,02005378	0,64172109
1-2-Cross-Exchange	1001	96	0,00263426	0,0337712	12,81999499	1,34874112	0,04214816	1,31496992	0,00837696	0,02503905	0,80124969
1-3-Cross-Exchange	101	320	0,00039931	0,00034443	0,86256292	0,20444672	0,00638896	0,20410229	0,00604453	0,00168469	0,05391018
1-3-Cross-Exchange	256	96	0,00086037	0,00210659	2,44846984	0,44050944	0,01376592	0,43840285	0,01165933	0,00478217	0,15302937
1-3-Cross-Exchange	301	416	0,00095923	0,00287128	2,99331756	0,49112576	0,01534768	0,48825448	0,0124764	0,00584632	0,18708235
1-3-Cross-Exchange	324	160	0,00097295	0,00345203	3,54800349	0,4981504	0,0155672	0,49469837	0,01211517	0,00692969	0,22175022
1-3-Cross-Exchange	400	128	0,00106018	0,0052796	4,97990907	0,54281216	0,01696288	0,53753256	0,01168328	0,00972638	0,31124432
1-3-Cross-Exchange	484	352	0,00107905	0,00719959	6,67215606	0,5524736	0,0172648	0,54527401	0,01006521	0,01303155	0,41700975
1-3-Cross-Exchange	601	320	0,00139731	0,01051596	7,5258604	0,71542272	0,02235696	0,70490676	0,011841	0,01469895	0,47036628
1-3-Cross-Exchange	801	96	0,00192441	0,01838195	9,55199256	0,98529792	0,03079056	0,96691597	0,01240861	0,01865624	0,59699953
1-3-Cross-Exchange	1001	96	0,00252551	0,02947868	11,67236717	1,29306112	0,04040816	1,26358244	0,01092948	0,02279759	0,72952295
1-4-Cross-Exchange	101	640	0,00039925	0,00030544	0,76503444	0,204416	0,006388	0,20411056	0,00608256	0,00149421	0,04781465
1-4-Cross-Exchange	256	96	0,00086114	0,00209325	2,43078942	0,44090368	0,01377824	0,43881043	0,01168499	0,00474764	0,15192434
1-4-Cross-Exchange	301	416	0,00095358	0,00273636	2,86956522	0,48823296	0,01525728	0,4854966	0,01252092	0,00560462	0,17934783
1-4-Cross-Exchange	324	160	0,00097359	0,00347788	3,57222239	0,49847808	0,01557744	0,4950002	0,01209956	0,006977	0,2232639
1-4-Cross-Exchange	400	128	0,00105895	0,00541968	5,11797535	0,5421824	0,0169432	0,53676272	0,01152352	0,00999605	0,31987346
1-4-Cross-Exchange	484	352	0,00108375	0,00739597	6,82442445	0,55488	0,01734	0,54748403	0,00994403	0,01332895	0,42652653
1-4-Cross-Exchange	601	96	0,00139511	0,01052405	7,54352703	0,71429632	0,02232176	0,70377227	0,01179771	0,01473345	0,47147044
1-4-Cross-Exchange	801	96	0,00183358	0,01864537	10,16883365	0,93879296	0,02933728	0,92014759	0,01069191	0,019861	0,6355521
1-4-Cross-Exchange	1001	96	0,00240112	0,02981043	12,41521873	1,22937344	0,03841792	1,19956301	0,00860749	0,02424847	0,77595117
1-5-Cross-Exchange	101	608	0,00039922	0,00030319	0,75945594	0,20440064	0,00638752	0,20409745	0,00608433	0,00148331	0,047466
1-5-Cross-Exchange	256	96	0,00087056	0,00218262	2,50714483	0,44572672	0,01392896	0,4435441	0,01174634	0,00489677	0,15669655
1-5-Cross-Exchange	301	416	0,00097319	0,00274873	2,8244536	0,49827328	0,01557104	0,49552455	0,01282231	0,00551651	0,17652835
1-5-Cross-Exchange	324	160	0,00096131	0,00365225	3,7992427	0,49219072	0,01538096	0,48853847	0,01172871	0,0074204	0,23745267
1-5-Cross-Exchange	400	128	0,00107255	0,00566555	5,28231784	0,5491456	0,0171608	0,54348005	0,01149525	0,01031703	0,33014487
1-5-Cross-Exchange	484	352	0,0010939	0,00787001	7,19445105	0,5600768	0,0175024	0,55220679	0,00963239	0,01405166	0,44965319
1-5-Cross-Exchange	601	96	0,00138356	0,01098304	7,93824626	0,70838272	0,02213696	0,69739968	0,01115392	0,01550439	0,49614039
1-5-Cross-Exchange	801	96	0,00175933	0,01964543	11,16642699	0,90077696	0,02814928	0,88113153	0,00850385	0,02180943	0,69790169
1-5-Cross-Exchange	1001	96	0,00234982	0,03125893	13,30269127	1,20310784	0,03759712	1,17184891	0,00633819	0,02598182	0,8314182
2-1-Cross-Exchange	101	320	0,00039782	0,00040596	1,02046152	0,20368384	0,00636512	0,20327788	0,00595916	0,00199309	0,06377884

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
2-1-Cross-Exchange	256	128	0,00086938	0,00250221	2,87815455	0,44512256	0,01391008	0,44262035	0,01140787	0,0056214	0,17988466
2-1-Cross-Exchange	301	416	0,00095801	0,0032103	3,35100886	0,49050112	0,01532816	0,48729082	0,01211786	0,00654494	0,20943805
2-1-Cross-Exchange	324	160	0,00096647	0,00391782	4,05374197	0,49483264	0,01546352	0,49091482	0,0115457	0,00791746	0,25335887
2-1-Cross-Exchange	400	128	0,00107186	0,00588641	5,49177131	0,54879232	0,01714976	0,54290591	0,01126335	0,01072612	0,34323571
2-1-Cross-Exchange	484	352	0,00107399	0,00808246	7,52563804	0,54988288	0,01718384	0,54180042	0,00910138	0,01469851	0,47035238
2-1-Cross-Exchange	601	320	0,00137978	0,01193531	8,65015437	0,70644736	0,02207648	0,69451205	0,01014117	0,01689483	0,54063465
2-1-Cross-Exchange	801	352	0,00186718	0,02087529	11,18011654	0,95599616	0,02987488	0,93512087	0,00899959	0,02183617	0,69875728
2-1-Cross-Exchange	1001	96	0,00256406	0,03443842	13,43120676	1,31279872	0,04102496	1,2783603	0,00658654	0,02623283	0,83945042
2-2-Cross-Exchange	101	320	0,00039598	0,00043431	1,09679782	0,20274176	0,00633568	0,20230745	0,00590137	0,00214218	0,06854986
2-2-Cross-Exchange	256	96	0,00086037	0,00267801	3,11262596	0,44050944	0,01376592	0,43783143	0,01108791	0,00607935	0,19453912
2-2-Cross-Exchange	301	416	0,00095478	0,00387208	4,05546828	0,48884736	0,01527648	0,48497528	0,0114044	0,00792084	0,25346677
2-2-Cross-Exchange	324	160	0,00095628	0,00431577	4,51308194	0,48961536	0,01530048	0,48529959	0,01098471	0,00881461	0,28206762
2-2-Cross-Exchange	400	128	0,0010588	0,00654715	6,18355686	0,5421056	0,0169408	0,53555845	0,01039365	0,01207726	0,3864723
2-2-Cross-Exchange	484	352	0,00108022	0,00923174	8,54616652	0,55307264	0,01728352	0,5438409	0,00805178	0,01669173	0,53413541
2-2-Cross-Exchange	601	320	0,00138046	0,01366672	9,90012025	0,70679552	0,02208736	0,6931288	0,00842064	0,01933617	0,61875752
2-2-Cross-Exchange	801	96	0,00190964	0,02335605	12,23060367	0,97773568	0,03055424	0,95437963	0,00719819	0,0238879	0,76441273
2-2-Cross-Exchange	1001	96	0,00249128	0,03998545	16,05016297	1,27553536	0,03986048	1,23554991	-0,00012497	0,03134797	1,00313519
2-3-Cross-Exchange	101	320	0,00039739	0,00039103	0,98399557	0,20346368	0,00635824	0,20307265	0,00596721	0,00192187	0,06149972
2-3-Cross-Exchange	256	128	0,00086918	0,00261831	3,01239099	0,44502016	0,01390688	0,44240185	0,01128857	0,00588358	0,18827444
2-3-Cross-Exchange	301	416	0,0009643	0,00335271	3,47683294	0,4937216	0,0154288	0,49036889	0,01207609	0,00679069	0,21730206
2-3-Cross-Exchange	324	160	0,00096497	0,00410521	4,25423588	0,49406464	0,01543952	0,48995943	0,01133431	0,00830905	0,26588974
2-3-Cross-Exchange	400	128	0,00106524	0,00630352	5,91746461	0,54540288	0,01704384	0,53909936	0,01074032	0,01155755	0,36984154
2-3-Cross-Exchange	484	352	0,00107785	0,00879667	8,16131187	0,5518592	0,0172456	0,54306253	0,00844893	0,01594006	0,51008199
2-3-Cross-Exchange	601	96	0,00137524	0,01253705	9,11626334	0,70412288	0,02200384	0,69158583	0,00946679	0,0178052	0,56976646
2-3-Cross-Exchange	801	96	0,00184852	0,02202168	11,91314132	0,94644224	0,02957632	0,92442056	0,00755464	0,02326785	0,74457133
2-3-Cross-Exchange	1001	96	0,00243698	0,0363841	14,92999532	1,24773376	0,03899168	1,21134966	0,00260758	0,02916015	0,93312471
2-4-Cross-Exchange	101	608	0,00039603	0,0003435	0,86735853	0,20276736	0,00633648	0,20242386	0,00599298	0,00169406	0,05420991
2-4-Cross-Exchange	256	128	0,00086708	0,0024774	2,85717581	0,44394496	0,01387328	0,44146756	0,01139588	0,00558042	0,17857349
2-4-Cross-Exchange	301	416	0,00096416	0,00308805	3,20283978	0,49364992	0,01542656	0,49056187	0,01233851	0,00625555	0,20017749
2-4-Cross-Exchange	324	160	0,00096418	0,0039625	4,10970981	0,49366016	0,01542688	0,48969766	0,01146438	0,00802678	0,25685686
2-4-Cross-Exchange	400	128	0,00105416	0,0061565	5,84019504	0,53972992	0,01686656	0,53357342	0,01071006	0,01140663	0,36501219
2-4-Cross-Exchange	484	352	0,0010867	0,00853266	7,85190025	0,5563904	0,0173872	0,54785774	0,00885454	0,01533574	0,49074377
2-4-Cross-Exchange	601	96	0,00135846	0,01167586	8,59492366	0,69553152	0,02173536	0,68385566	0,0100595	0,01678696	0,53718273
2-4-Cross-Exchange	801	96	0,00179368	0,02069778	11,53928237	0,91836416	0,02869888	0,89766638	0,0080011	0,02253766	0,72120515
2-4-Cross-Exchange	1001	96	0,00236163	0,03309784	14,01482874	1,20915456	0,03778608	1,17605672	0,00468824	0,02737271	0,8759268
2-5-Cross-Exchange	101	320	0,00039508	0,00030724	0,77766528	0,20228096	0,00632128	0,20197372	0,00601404	0,00151888	0,04860408
2-5-Cross-Exchange	256	128	0,00086017	0,00243577	2,83173094	0,44040704	0,01376272	0,43797127	0,01132695	0,00553072	0,17698318
2-5-Cross-Exchange	301	416	0,00096567	0,00290612	3,00943386	0,49442304	0,01545072	0,49151692	0,0125446	0,0058778	0,18808962
2-5-Cross-Exchange	324	640	0,00097048	0,0039293	4,0488212	0,49688576	0,01552768	0,49295646	0,01159838	0,00790785	0,25305133
2-5-Cross-Exchange	400	128	0,00105924	0,00618554	5,83960198	0,54233088	0,01694784	0,53614534	0,0107623	0,01140547	0,36497512
2-5-Cross-Exchange	484	352	0,00108683	0,00861843	7,92987864	0,55645696	0,01738928	0,54783853	0,00877085	0,01548804	0,49561741
2-5-Cross-Exchange	601	96	0,00134181	0,01142522	8,51478227	0,68700672	0,02146896	0,6755815	0,01004374	0,01663043	0,53217389
2-5-Cross-Exchange	801	96	0,00174906	0,02035168	11,63578151	0,89551872	0,02798496	0,87516704	0,00763328	0,0272614	0,72723634
2-5-Cross-Exchange	1001	96	0,00229887	0,0322858	14,04420433	1,17702144	0,03678192	1,14473564	0,00449612	0,02743009	0,87776277
3-1-Cross-Exchange	101	320	0,00039655	0,00033909	0,85510024	0,2030336	0,0063448	0,20269451	0,00600571	0,00167012	0,05344376
3-1-Cross-Exchange	256	128	0,00086664	0,00227737	2,62781547	0,44371968	0,01386624	0,44144231	0,01158887	0,00513245	0,16423847
3-1-Cross-Exchange	301	416	0,00096716	0,00285618	2,95316183	0,49518592	0,01547456	0,49232974	0,01261838	0,0057689	0,18457261
3-1-Cross-Exchange	324	160	0,00096347	0,00360429	3,74094679	0,49329664	0,01541552	0,48969235	0,01181123	0,00730654	0,23380917
3-1-Cross-Exchange	400	128	0,00105914	0,00550483	5,19745265	0,54227968	0,01694624	0,53677485	0,01144141	0,01015127	0,32484079
3-1-Cross-Exchange	484	352	0,0010662	0,00746417	7,00072219	0,5458944	0,0170592	0,53843023	0,00959503	0,01367329	0,43754514
3-1-Cross-Exchange	601	320	0,00133408	0,01053488	7,89673783	0,68304896	0,02134528	0,67251408	0,0108104	0,01542332	0,49354611
3-1-Cross-Exchange	801	352	0,00181466	0,0184844	10,18615057	0,92910592	0,02903456	0,91062152	0,01055016	0,01989483	0,63663441
3-1-Cross-Exchange	1001	96	0,00241721	0,02942371	12,17259154	1,23761152	0,03867536	1,20818781	0,00925165	0,02377459	0,76078697
3-2-Cross-Exchange	101	320	0,000394	0,00038702	0,98228426	0,201728	0,006304	0,20134098	0,00591698	0,00191852	0,06139277
3-2-Cross-Exchange	256	128	0,00085615	0,00264049	3,08414413	0,4383488	0,0136984	0,43570831	0,01105791	0,00602372	0,19275901

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
3-2-Cross-Exchange	301	416	0,00095704	0,00331714	3,46604113	0,49000448	0,01531264	0,48668734	0,0119955	0,00676961	0,21662757
3-2-Cross-Exchange	324	160	0,00096604	0,00418335	4,33041075	0,49461248	0,01545664	0,49042913	0,01127329	0,00845783	0,27065067
3-2-Cross-Exchange	400	128	0,00106702	0,00633014	5,9325411	0,54631424	0,01707232	0,5399841	0,01074218	0,01158699	0,37078382
3-2-Cross-Exchange	484	352	0,00108171	0,00886227	8,19283357	0,55383552	0,01730736	0,54497325	0,00844509	0,01600163	0,5120521
3-2-Cross-Exchange	601	320	0,00134189	0,01244371	9,27327128	0,68704768	0,02147024	0,67460397	0,00902653	0,01811186	0,57957946
3-2-Cross-Exchange	801	96	0,00180132	0,02187116	12,1417405	0,92227584	0,02882112	0,90040468	0,00694996	0,02371434	0,75885878
3-2-Cross-Exchange	1001	96	0,00240003	0,03607282	15,03015379	1,22881536	0,03840048	1,19274254	0,00232766	0,02935577	0,93938461
3-3-Cross-Exchange	101	320	0,00039572	0,00038186	0,96497524	0,20260864	0,00633152	0,20222678	0,00594966	0,00188472	0,06031095
3-3-Cross-Exchange	256	96	0,00086295	0,00270788	3,13793383	0,4418304	0,0138072	0,43912252	0,01109932	0,00612878	0,19612086
3-3-Cross-Exchange	301	416	0,00096355	0,00359863	3,73476208	0,4933376	0,0154168	0,48973897	0,01181817	0,00729446	0,23342263
3-3-Cross-Exchange	324	160	0,00097173	0,004429	4,55785043	0,49752576	0,01554768	0,49309676	0,01111868	0,00890205	0,28486565
3-3-Cross-Exchange	400	128	0,00106421	0,00673434	6,32801797	0,54487552	0,01702736	0,53814118	0,01029302	0,01235941	0,39550112
3-3-Cross-Exchange	484	352	0,00107915	0,00954754	8,84727795	0,5525248	0,0172664	0,54297726	0,00771886	0,01727984	0,55295487
3-3-Cross-Exchange	601	320	0,00134672	0,01318825	9,79286711	0,68952064	0,02154752	0,67633239	0,00835927	0,01912669	0,61205419
3-3-Cross-Exchange	801	96	0,0017994	0,02302071	12,79354785	0,9212928	0,0287904	0,89827209	0,00576969	0,0249874	0,79959674
3-3-Cross-Exchange	1001	96	0,00234491	0,03890452	16,5910504	1,20059392	0,03751856	1,1616894	-0,00138596	0,0324044	1,03694065
3-4-Cross-Exchange	101	320	0,00039507	0,00034791	0,88062875	0,20227584	0,00632112	0,20192793	0,00597321	0,00171998	0,0550393
3-4-Cross-Exchange	256	96	0,00086027	0,00260908	3,03286178	0,44045824	0,01376432	0,43784916	0,01115524	0,00592356	0,18955386
3-4-Cross-Exchange	301	416	0,00097512	0,00328746	3,37133891	0,49926144	0,01560192	0,49597398	0,01231446	0,00658465	0,21070868
3-4-Cross-Exchange	324	160	0,0009753	0,0043212	4,43063673	0,4993536	0,0156048	0,4950324	0,0112836	0,00865359	0,2769148
3-4-Cross-Exchange	400	128	0,00106954	0,00666078	6,22770537	0,54760448	0,01711264	0,5409437	0,01045186	0,01216349	0,38923159
3-4-Cross-Exchange	484	352	0,00107873	0,00937863	8,69414033	0,55230976	0,01725968	0,54293113	0,00788105	0,01698074	0,54338377
3-4-Cross-Exchange	601	96	0,00135027	0,01253368	9,28235094	0,69133824	0,02160432	0,67880456	0,00907064	0,01812959	0,58014693
3-4-Cross-Exchange	801	96	0,00178134	0,02220076	12,46295485	0,91204608	0,02850144	0,88984532	0,00630068	0,02434171	0,77893468
3-4-Cross-Exchange	1001	96	0,0022903	0,03646274	15,92050823	1,1726336	0,0366448	1,13617086	0,00018206	0,03109474	0,99503176
3-5-Cross-Exchange	101	320	0,00039526	0,00030816	0,77963872	0,20237312	0,00632416	0,20206496	0,006016	0,00152273	0,04872742
3-5-Cross-Exchange	256	128	0,00085452	0,0026058	3,04943126	0,43751424	0,01367232	0,43490844	0,01106652	0,00595592	0,19058945
3-5-Cross-Exchange	301	416	0,0009719	0,00303848	3,12632987	0,4976128	0,0155504	0,49457432	0,01251192	0,00610611	0,19539562
3-5-Cross-Exchange	324	640	0,00097244	0,00416234	4,28030521	0,49788928	0,01555904	0,49372694	0,0113967	0,00835997	0,26751908
3-5-Cross-Exchange	400	288	0,00107182	0,00647521	6,04132224	0,54877184	0,01714912	0,54229663	0,01067391	0,01179946	0,37758264
3-5-Cross-Exchange	484	352	0,00108761	0,0093269	8,57559235	0,55685632	0,01740176	0,54752942	0,00807486	0,0167492	0,53597452
3-5-Cross-Exchange	601	96	0,0013397	0,01180191	8,80936777	0,6859264	0,0214352	0,67412449	0,00963329	0,0172058	0,55058549
3-5-Cross-Exchange	801	96	0,00175368	0,02102214	11,98744355	0,89788416	0,02805888	0,87686202	0,00703674	0,02341298	0,74921522
3-5-Cross-Exchange	1001	96	0,00225717	0,03353756	14,85823398	1,15567104	0,03611472	1,12213348	0,00257716	0,02901999	0,92863962
4-1-Cross-Exchange	101	160	0,00039637	0,00029823	0,75240306	0,20294144	0,00634192	0,20264321	0,00604369	0,00146954	0,04702519
4-1-Cross-Exchange	256	96	0,00086158	0,00214628	2,49109775	0,44112896	0,01378528	0,43898268	0,011639	0,00486543	0,15569361
4-1-Cross-Exchange	301	416	0,00095324	0,0026732	2,80433049	0,48805888	0,01525184	0,48538568	0,01257864	0,00547721	0,17527066
4-1-Cross-Exchange	324	160	0,00096551	0,00354322	3,66979109	0,49434112	0,01544816	0,4907979	0,01190494	0,00716756	0,22936194
4-1-Cross-Exchange	400	128	0,00105797	0,00548818	5,18746278	0,54168064	0,01692752	0,53619246	0,01143934	0,01013176	0,32421642
4-1-Cross-Exchange	484	352	0,00107428	0,00749393	6,97576982	0,55003136	0,01718848	0,54253743	0,00969455	0,01362455	0,43598561
4-1-Cross-Exchange	601	320	0,00129738	0,01025142	7,90163252	0,66425856	0,02075808	0,65400714	0,01050666	0,01543288	0,49385203
4-1-Cross-Exchange	801	352	0,00177123	0,01830961	10,33722893	0,90686976	0,02833968	0,88856015	0,01003007	0,0201899	0,64607681
4-1-Cross-Exchange	1001	96	0,00231327	0,02882727	12,46169708	1,18439424	0,03701232	1,15556697	0,00818505	0,02433925	0,77885607
4-2-Cross-Exchange	101	320	0,00039627	0,0003261	0,82292376	0,20289024	0,00634032	0,20256414	0,00601422	0,00160727	0,05143274
4-2-Cross-Exchange	256	96	0,00085991	0,00239238	2,78212836	0,44027392	0,01375856	0,43788154	0,01136618	0,00543384	0,17388302
4-2-Cross-Exchange	301	416	0,00095279	0,00297691	3,12441356	0,48782848	0,01524464	0,48485157	0,01226773	0,00610237	0,19527585
4-2-Cross-Exchange	324	160	0,00096024	0,00393959	4,1027139	0,49164288	0,01536384	0,48770329	0,01142425	0,00801311	0,25641962
4-2-Cross-Exchange	400	128	0,00105138	0,00609287	5,79511689	0,53830656	0,01682208	0,53221369	0,01072921	0,01131859	0,36219481
4-2-Cross-Exchange	484	352	0,00108234	0,00848976	7,84389379	0,55415808	0,01731744	0,54566832	0,00882768	0,01532011	0,49024336
4-2-Cross-Exchange	601	320	0,00131986	0,01121233	8,49509039	0,67576832	0,02111776	0,66455599	0,00990543	0,01659197	0,53094315
4-2-Cross-Exchange	801	96	0,00170883	0,01980167	11,58785251	0,87492096	0,02734128	0,85511929	0,00753961	0,02263252	0,72424078
4-2-Cross-Exchange	1001	96	0,00226993	0,03170781	13,96862899	1,16220416	0,03631888	1,13049635	0,00461107	0,02728248	0,87303931
4-3-Cross-Exchange	101	320	0,00039427	0,00033943	0,8609075	0,20186624	0,00630832	0,20152681	0,00596889	0,00168146	0,05380672
4-3-Cross-Exchange	256	96	0,00085707	0,0026369	3,07664485	0,43881984	0,01371312	0,43618294	0,01107622	0,00600907	0,1922903
4-3-Cross-Exchange	301	416	0,00095377	0,00323183	3,3884794	0,48833024	0,01526032	0,48509841	0,01202849	0,00661812	0,21177996

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
4-3-Cross-Exchange	324	160	0,00096435	0,0043348	4,49504848	0,4937472	0,0154296	0,4894124	0,0110948	0,00877939	0,28094053
4-3-Cross-Exchange	400	128	0,00105244	0,00668669	6,35351184	0,53884928	0,01683904	0,53216259	0,01015235	0,0124092	0,39709449
4-3-Cross-Exchange	484	352	0,00109179	0,00940726	8,61636395	0,55899648	0,01746864	0,54958922	0,00806138	0,01682884	0,53852275
4-3-Cross-Exchange	601	96	0,00130967	0,01224425	9,34911084	0,67055104	0,02095472	0,65830679	0,00871047	0,01825998	0,58431943
4-3-Cross-Exchange	801	96	0,00171159	0,02181023	12,74267202	0,87633408	0,02738544	0,85452385	0,00557521	0,02488803	0,796417
4-3-Cross-Exchange	1001	96	0,00222619	0,03585759	16,10715617	1,13980928	0,03561904	1,10395169	-0,00023855	0,03145929	1,00669726
4-4-Cross-Exchange	101	320	0,00039604	0,00033032	0,83405717	0,20277248	0,00633664	0,20244216	0,00600632	0,00162902	0,05212857
4-4-Cross-Exchange	256	96	0,00086019	0,00273514	3,17969286	0,44041728	0,01376304	0,43768214	0,0110279	0,00621034	0,1987308
4-4-Cross-Exchange	301	416	0,00095149	0,00338375	3,55626439	0,48716288	0,01522384	0,48377913	0,01184009	0,00694583	0,22226652
4-4-Cross-Exchange	324	160	0,00096197	0,00452932	4,70837968	0,49252864	0,01539152	0,48799932	0,0108622	0,00919605	0,29427373
4-4-Cross-Exchange	400	128	0,00106178	0,00693728	6,5336322	0,54363136	0,01698848	0,53669408	0,0100512	0,012761	0,40835201
4-4-Cross-Exchange	484	352	0,00108844	0,00992241	9,11617544	0,55728128	0,01741504	0,54735887	0,00749263	0,01780503	0,56976097
4-4-Cross-Exchange	601	96	0,00130909	0,01265439	9,66655463	0,67025408	0,02094544	0,65759969	0,00829105	0,01887999	0,60415966
4-4-Cross-Exchange	801	96	0,00172489	0,0224343	13,00622069	0,88314368	0,02759824	0,86070938	0,00516394	0,02540277	0,81288879
4-4-Cross-Exchange	1001	96	0,00219677	0,03698529	16,83621408	1,12474624	0,03514832	1,08776095	-0,00183697	0,03288323	1,05226338
4-5-Cross-Exchange	101	320	0,00039503	0,0002991	0,75715768	0,20225536	0,00632048	0,20195626	0,00602138	0,00147882	0,04732236
4-5-Cross-Exchange	256	96	0,00085996	0,0026696	3,10433043	0,44029952	0,01375936	0,43762992	0,01108976	0,00606315	0,19402065
4-5-Cross-Exchange	301	416	0,00096349	0,0031135	3,23148139	0,49330688	0,01541584	0,49019338	0,01230234	0,00631149	0,20196759
4-5-Cross-Exchange	324	160	0,00096573	0,00447651	4,63536392	0,49445376	0,01545168	0,48997725	0,01097517	0,00905345	0,28971025
4-5-Cross-Exchange	400	128	0,00106328	0,00695877	6,54462606	0,54439936	0,01701248	0,53744059	0,01005371	0,01278247	0,40903913
4-5-Cross-Exchange	484	352	0,00108648	0,00991666	9,12732862	0,55627776	0,01738368	0,5463611	0,00746702	0,01782681	0,57045804
4-5-Cross-Exchange	601	96	0,00131777	0,01213738	9,21054509	0,67469824	0,02108432	0,66256086	0,00894694	0,01798935	0,57565907
4-5-Cross-Exchange	801	96	0,00170277	0,02174245	12,76887072	0,87181824	0,02724432	0,85007579	0,00550187	0,0249392	0,79805442
4-5-Cross-Exchange	1001	96	0,00218705	0,0353578	16,16689147	1,1197696	0,0349928	1,0844118	-0,000365	0,03157596	1,01043072
5-1-Cross-Exchange	101	320	0,00039296	0,00026975	0,68645664	0,20119552	0,00628736	0,20092577	0,00601761	0,00134074	0,04290354
5-1-Cross-Exchange	256	96	0,00085814	0,00210275	2,45035775	0,43936768	0,01373024	0,43726493	0,01162749	0,00478585	0,15314736
5-1-Cross-Exchange	301	416	0,00094681	0,00252486	2,66670187	0,48476672	0,01514896	0,48224186	0,0126241	0,0052084	0,16666887
5-1-Cross-Exchange	324	160	0,00095706	0,00353964	3,69845151	0,49001472	0,01531296	0,48647508	0,01177332	0,00722354	0,23115322
5-1-Cross-Exchange	400	128	0,00104964	0,0054909	5,23122213	0,53741568	0,01679424	0,53192478	0,01130334	0,01021723	0,32695138
5-1-Cross-Exchange	484	352	0,00107447	0,00758801	7,06209573	0,55012864	0,01719152	0,54254063	0,00960351	0,01379316	0,44138098
5-1-Cross-Exchange	601	320	0,00127644	0,00997931	7,81807997	0,65353728	0,02042304	0,64355797	0,01044373	0,01526969	0,48863
5-1-Cross-Exchange	801	352	0,00170205	0,01800906	10,5808055	0,8714496	0,0272328	0,85344054	0,00922374	0,02066564	0,66130034
5-1-Cross-Exchange	1001	96	0,0021946	0,02830143	12,89594003	1,1236352	0,0351136	1,09533377	0,00681217	0,02518738	0,80599625
5-2-Cross-Exchange	101	320	0,00039163	0,00028429	0,72591477	0,20051456	0,00626608	0,20023027	0,00598179	0,0014178	0,04536967
5-2-Cross-Exchange	256	96	0,00085874	0,00230434	2,6833966	0,43967488	0,01373984	0,43737054	0,0114355	0,00524101	0,16771229
5-2-Cross-Exchange	301	416	0,00095579	0,00270634	2,83152157	0,48936448	0,01529264	0,48665814	0,0125863	0,00553032	0,1769701
5-2-Cross-Exchange	324	160	0,00095477	0,00385576	4,03841763	0,48884224	0,01527632	0,48498648	0,01142056	0,00788753	0,2524011
5-2-Cross-Exchange	400	128	0,00105561	0,00600533	5,68896657	0,54047232	0,01688976	0,53446699	0,01088443	0,01111126	0,35556041
5-2-Cross-Exchange	484	352	0,00108051	0,00833902	7,71767036	0,55322112	0,01728816	0,5448821	0,00894914	0,01507357	0,4823544
5-2-Cross-Exchange	601	320	0,00128596	0,01051141	8,17397897	0,65841152	0,02057536	0,64790011	0,01006395	0,0159648	0,51087369
5-2-Cross-Exchange	801	96	0,00167227	0,01869121	11,17714843	0,85620224	0,02675632	0,83751103	0,00806511	0,02183037	0,69857178
5-2-Cross-Exchange	1001	96	0,00215842	0,02950308	13,66883183	1,10511104	0,03453472	1,07560796	0,00503164	0,02669694	0,85430199
5-3-Cross-Exchange	101	320	0,00039136	0,00029359	0,75017886	0,20037632	0,00626176	0,20008273	0,00596817	0,0146519	0,04688618
5-3-Cross-Exchange	256	96	0,00085571	0,00252387	2,94944549	0,43812352	0,01369136	0,43559965	0,01116749	0,00576064	0,18434034
5-3-Cross-Exchange	301	416	0,00094992	0,00293826	3,09316574	0,48635904	0,01519872	0,48342078	0,01226046	0,00604134	0,19332286
5-3-Cross-Exchange	324	160	0,00096789	0,00420696	4,34652698	0,49555968	0,01548624	0,49135272	0,01127928	0,00848931	0,27165794
5-3-Cross-Exchange	400	128	0,00104844	0,00656974	6,26620503	0,53680128	0,01677504	0,53023154	0,0102053	0,01223868	0,39163781
5-3-Cross-Exchange	484	352	0,001076	0,00925	8,59665428	0,550912	0,017216	0,541662	0,007966	0,01679034	0,53729089
5-3-Cross-Exchange	601	320	0,00127684	0,01129878	8,84901789	0,65374208	0,02042944	0,6424433	0,00913066	0,01728324	0,55306362
5-3-Cross-Exchange	801	96	0,00164278	0,0201265	12,25148833	0,84110336	0,02628448	0,82097686	0,00615798	0,02392869	0,76571802
5-3-Cross-Exchange	1001	96	0,00212098	0,03207376	15,12214165	1,08594176	0,03393568	1,053868	0,00186192	0,02953543	0,94513385
5-4-Cross-Exchange	101	320	0,00039401	0,00029191	0,74086952	0,20173312	0,00630416	0,20144121	0,00601225	0,00144701	0,04630435
5-4-Cross-Exchange	256	96	0,00085288	0,00266336	3,12278398	0,43667456	0,01364608	0,4340112	0,01098272	0,00609919	0,195174
5-4-Cross-Exchange	301	416	0,000968	0,00304221	3,14277893	0,495616	0,015488	0,49257379	0,01244579	0,00613824	0,19642368
5-4-Cross-Exchange	324	160	0,00096529	0,00445849	4,61880886	0,49422848	0,01544464	0,48976999	0,01098615	0,00902111	0,28867555

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
5-4-Cross-Exchange	400	128	0,00104947	0,00690472	6,57924476	0,53732864	0,01679152	0,53042392	0,0098868	0,01285009	0,4112028
5-4-Cross-Exchange	484	352	0,00107393	0,00981391	9,13831442	0,54985216	0,01718288	0,54003825	0,00736897	0,01784827	0,57114465
5-4-Cross-Exchange	601	96	0,00129224	0,01178854	9,12256237	0,66162688	0,02067584	0,64983834	0,0088873	0,0178175	0,57016015
5-4-Cross-Exchange	801	96	0,00165625	0,02109996	12,73959849	0,848	0,0265	0,82690004	0,00540004	0,02488203	0,79622491
5-4-Cross-Exchange	1001	96	0,00210979	0,03449355	16,34928121	1,08021248	0,03375664	1,04571893	-0,00073691	0,03193219	1,02183008
5-5-Cross-Exchange	101	320	0,00039517	0,00027647	0,69962295	0,20232704	0,00632272	0,20205057	0,00604625	0,00136645	0,04372643
5-5-Cross-Exchange	256	96	0,00086323	0,00274146	3,17581641	0,44197376	0,01381168	0,4392323	0,01107022	0,00620277	0,19848853
5-5-Cross-Exchange	301	416	0,0009584	0,00310078	3,23537145	0,4907008	0,0153344	0,48760002	0,01223362	0,00631908	0,20221072
5-5-Cross-Exchange	324	160	0,00096892	0,00460454	4,75223961	0,49608704	0,01550272	0,4914825	0,01089818	0,00928172	0,29701498
5-5-Cross-Exchange	400	128	0,00104805	0,00711979	6,79336864	0,5366016	0,0167688	0,52948181	0,00964901	0,0132683	0,42458554
5-5-Cross-Exchange	484	352	0,00108427	0,01026942	9,4712756	0,55514624	0,01734832	0,54487682	0,0070789	0,01849859	0,59195473
5-5-Cross-Exchange	601	320	0,00128401	0,01191939	9,28294172	0,65741312	0,02054416	0,64549373	0,00862477	0,01813075	0,58018386
5-5-Cross-Exchange	801	96	0,00166609	0,02125956	12,76015101	0,85303808	0,02665744	0,83177852	0,00539788	0,02492217	0,79750944
5-5-Cross-Exchange	1001	96	0,00210294	0,03458796	16,44743074	1,07670528	0,03364704	1,04211732	-0,00094092	0,03212389	1,02796442
2-Opt	101	320	0,00039435	0,00016507	0,41858755	0,2019072	0,0063096	0,20174213	0,00614453	0,00081755	0,02616172
2-Opt	256	96	0,0008302	0,00076612	0,92281378	0,4250624	0,0132832	0,42429628	0,01251708	0,00180237	0,05767586
2-Opt	301	480	0,00093593	0,00117819	1,25884414	0,47919616	0,01497488	0,47801797	0,01379669	0,00245868	0,07867776
2-Opt	324	160	0,00093162	0,00123681	1,32759065	0,47698944	0,01490592	0,47575263	0,01366911	0,00259295	0,08297442
2-Opt	400	128	0,00097521	0,00179168	1,8372248	0,49930752	0,01560336	0,49751584	0,01381168	0,00358833	0,11482655
2-Opt	484	352	0,00101405	0,00253027	2,49521227	0,5191936	0,0162248	0,51666333	0,01369453	0,00487346	0,15595077
2-Opt	601	288	0,00111776	0,00446715	3,99651983	0,57229312	0,01788416	0,56782597	0,01341701	0,0078057	0,24978249
2-Opt	801	352	0,00132632	0,0078238	5,8988781	0,67907584	0,02122112	0,67125204	0,01339732	0,01152125	0,36867988
2-Opt	1001	96	0,00161846	0,01264911	7,81552216	0,82865152	0,02589536	0,81600241	0,01324625	0,01526469	0,48847014
2-Opt*	101	320	0,00039745	0,00016026	0,40322053	0,2034944	0,0063592	0,20333414	0,00619894	0,00078754	0,02520128
2-Opt*	256	128	0,00082041	0,00077282	0,94199242	0,42004992	0,01312656	0,4192771	0,01235374	0,00183983	0,05887453
2-Opt*	301	480	0,00093192	0,00116168	1,24654477	0,47714304	0,01491072	0,47598136	0,01374904	0,00243466	0,07790905
2-Opt*	324	640	0,00093155	0,00116244	1,24785572	0,4769536	0,0149048	0,47579116	0,01374236	0,00243722	0,07799098
2-Opt*	400	128	0,00100818	0,00173313	1,71906802	0,51618816	0,01613088	0,51445503	0,01439775	0,00335755	0,10744175
2-Opt*	484	352	0,00101686	0,00243948	2,39903232	0,52063232	0,01626976	0,51819284	0,01383028	0,00468561	0,14993952
2-Opt*	601	320	0,00117028	0,00443272	3,7877431	0,59918336	0,01872448	0,59475064	0,01429176	0,00739794	0,23673394
2-Opt*	801	352	0,00145273	0,00779558	5,36615889	0,74379776	0,02324368	0,73600218	0,0154481	0,01048078	0,33538493
2-Opt*	1001	96	0,0018467	0,01265101	6,85060378	0,9455104	0,0295472	0,93285939	0,01689619	0,01338009	0,42816274
Or-Opt	101	320	0,0003657	0,00024024	0,65693191	0,1872384	0,0058512	0,18699816	0,00561096	0,00128307	0,04105824
Or-Opt	256	96	0,00080462	0,00131255	1,63126693	0,41196544	0,01287392	0,41065289	0,01156137	0,00318607	0,10195418
Or-Opt	301	416	0,00086538	0,00179788	2,0775613	0,44307456	0,01384608	0,44127668	0,0120482	0,00405774	0,12984758
Or-Opt	324	160	0,00090987	0,00220704	2,4256652	0,46585344	0,01455792	0,4636464	0,01235088	0,00473763	0,15160408
Or-Opt	400	128	0,00099497	0,00320387	3,22006694	0,50942464	0,01591952	0,50622077	0,01271565	0,00628919	0,20125418
Or-Opt	484	352	0,00100016	0,00427754	4,2768557	0,51208192	0,01600256	0,50780438	0,01172502	0,00835323	0,26730348
Or-Opt	601	320	0,00125469	0,00681944	5,43515928	0,64240128	0,02007504	0,63558184	0,0132556	0,01061555	0,33969746
Or-Opt	801	352	0,00170737	0,01149392	6,73194445	0,87417344	0,02731792	0,86267952	0,015824	0,01314833	0,42074653
Or-Opt	1001	96	0,00234355	0,01835288	7,8312304	1,1998976	0,0374968	1,18154472	0,01914392	0,01529537	0,4894519
Relocate	101	320	0,00037452	0,00026925	0,71892022	0,19175424	0,00599232	0,19148499	0,00572307	0,00140414	0,04493251
Relocate	256	96	0,00078736	0,00136835	1,73789626	0,40312832	0,01259776	0,40175997	0,01122941	0,00339433	0,10861852
Relocate	301	480	0,00088072	0,00182953	2,07731174	0,45092864	0,01409152	0,44909911	0,01226199	0,00405725	0,12983198
Relocate	324	160	0,00089853	0,00217346	2,41890644	0,46004736	0,01437648	0,4578739	0,01220302	0,00472443	0,15118165
Relocate	400	128	0,00100294	0,00310443	3,09532973	0,51350528	0,01604704	0,51040085	0,01294261	0,00604557	0,19345811
Relocate	484	352	0,00097275	0,00393972	4,05008481	0,498048	0,015564	0,49410828	0,01162428	0,00791032	0,2531303
Relocate	601	320	0,0012933	0,00604519	4,67423645	0,6621696	0,0206928	0,65612441	0,01464761	0,00912937	0,29213978
Relocate	801	352	0,00166233	0,00961106	5,78167993	0,85111296	0,02659728	0,8415019	0,01698622	0,01129234	0,361355
Relocate	1001	32	0,0021557	0,0161357	7,48513244	1,1037184	0,0344912	1,0875827	0,0183555	0,0146194	0,46782078
Swap	101	320	0,00036702	0,00044382	1,2092529	0,18791424	0,00587232	0,18747042	0,0054285	0,00236182	0,07557831
Swap	256	96	0,00075749	0,0025053	3,30737039	0,38783488	0,01211984	0,38532958	0,00961454	0,00645971	0,20671065
Swap	301	416	0,00084459	0,00371207	4,39511479	0,43243008	0,01351344	0,42871801	0,00980137	0,00858421	0,27469467
Swap	324	160	0,00089585	0,00397298	4,43487191	0,4586752	0,0143336	0,45470222	0,01036062	0,00866186	0,27717949
Swap	400	128	0,00099583	0,00605523	6,08058604	0,50986496	0,01593328	0,50380973	0,00987805	0,01187614	0,38003663

Operator	Knoten	Threads	Mittelwert GPU	Mittelwert CPU	Speedup	Kosten p=512	Kosten p=16	Overhead p=512	Overhead p=16	Effizienz p=512	Effizienz p=16
Swap	484	416	0,00102017	0,00825183	8,0886813	0,52232704	0,01632272	0,51407521	0,00807089	0,01579821	0,50554258
Swap	601	320	0,00134072	0,01244756	9,28423534	0,68644864	0,02145152	0,67400108	0,00900396	0,01813327	0,58026471
Swap	801	352	0,00191734	0,02060931	10,74890734	0,98167808	0,03067744	0,96106877	0,01006813	0,02099396	0,67180671
Swap	1001	32	0,00263009	0,03558673	13,53061302	1,34660608	0,04208144	1,31101935	0,00649471	0,02642698	0,84566331

B.3 Lösungsverfahren der Literatur

Im Folgenden sind die Lösungsqualität und Zeitdauer der besten bekannten Lösungen der Verfahren der Literatur, die Ergebnisse im Rahmen der Golden-Instanzen liefern, dargestellt. Sie dienen als Referenz für die Auswertungen in Kapitel 4.2. Bei einigen Verfahren wird lediglich die benötigte Zeit durchschnittlich über alle Instanzen angegeben. Daraus folgt, dass für diese Instanzen die gleiche Rechenzeit für alle Instanzen angenommen wird.

Autoren	Abkürzung
Alabas-Uslu und Dengiz (2011)	AUD
Alba und Dorronsoro (2006)	AD
Chen et al. (2010)	CHD
Cordeau und Maischberger (2012)	CM
Ergun et al. (2006)	EOS
Groër et al. (2010)	GGWa
Groër et al. (2011)	GGWb
Ho und Gendreau (2006)	HG
Jin et al. (2012)	JCLa
Jin et al. (2011)	JCLb
Kwon et al. (2007)	KKSLK
Kytöjoki et al. (2007)	KNBG
Lee et al. (2010)	LLLY
Li et al. (2005)	LGW
Lin et al. (2009)	LLYL
Marinakis (2012)	M
Marinakis und Marinaki (2010)	MM
Mester und Bräysy (2007)	MB
Nagata (2007)	N
Nagata und Bräysy (2008)	NBa
Nagata und Bräysy (2009)	NBb
Prins (2004)	Pa
Prins (2009)	Pb
Reimann et al. (2004)	RDH
Szeto et al. (2011)	SWH
Tarantilis (2005)	Ta
Toth und Vigo (2003)	TV
Turky (2011)	Tb
Vidal et al. (2011)	VCGLR

Tabelle B.14: Abkürzungen der Lösungsverfahren der Literatur für CVRPs

Tabelle B.15: Ergebnisse von Lösungsverfahren der Literatur auf den Golden-Instanzen

Instanz	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit
	AUD		AD		CHD		CM		EOS		GGWa		GGWb	
g9	585,29	1144,93	589,20	-	585,24	22,03	582,95	1538,94	586,44	772,91	588,13	61,64	579,71	11668,07
g10	745,25	1144,93	754,95	-	745,57	22,03	739,95	1538,94	745,26	772,91	747,63	61,64	737,28	11668,07
g11	924,74	1144,93	935,91	-	923,13	22,03	920,54	1538,94	922,09	772,91	935,37	61,64	913,35	11668,07
g12	1123,29	1144,93	1126,38	-	1117,93	22,03	1111,76	1538,94	1121,15	772,91	1132,32	61,64	1102,76	11668,07
g13	861,94	1144,93	872,59	-	861,65	22,03	865,27	1538,94	869,26	772,91	868,13	61,64	857,19	11668,07
g14	1097,49	1144,93	1107,24	-	1088,87	22,03	1085,76	1538,94	1092,29	772,91	1093,69	61,64	1080,55	11668,07
g15	1356,34	1144,93	1391,27	-	1356,32	22,03	1355,86	1538,94	1357,97	772,91	1360,68	61,64	1338,19	11668,07
g16	1643,74	1144,93	1663,87	-	1637,13	22,03	1633,84	1538,94	1642,92	772,91	1639,24	61,64	1613,66	11668,07
g17	709,84	1144,93	714,73	-	708,89	22,03	708,17	1538,94	709,66	772,91	709,56	61,64	707,76	11668,07
g18	1005,97	1144,93	1019,19	-	1010,52	22,03	1000,86	1538,94	1010,70	772,91	1011,87	61,64	995,13	11668,07
g19	1387,93	1144,93	1367,47	-	1382,67	22,03	1373,07	1538,94	1385,84	772,91	1381,01	61,64	1365,60	11668,07
g20	1872,45	1144,93	1867,30	-	1839,93	22,03	1832,65	1538,94	1854,54	772,91	1842,63	61,64	1818,25	11668,07
Summe	13314,27	13739,11	13410,10	-	13257,86	264,38	13210,68	18467,28	13298,12	9274,87	13310,26	739,65	13109,43	140016,86

	HG		JCLa		JCLb		KKSLK		KNBG		LLLY		LGW	
g9	589,39	542,22	581,73	1883,80	580,17	227154,13	589,60	500,13	620,67	0,60	583,39	6358,18	588,25	10,66
g10	749,04	926,31	738,50	3893,51	737,18	227154,13	749,78	1412,20	784,77	1,31	766,55	6358,18	749,49	10,66
g11	927,99	1452,48	914,98	5939,50	913,30	227154,13	961,64	5739,67	986,80	1,39	946,61	6358,18	925,91	10,66
g12	1124,44	2083,40	1109,93	7337,74	1105,17	227154,13	1173,37	972,55	1209,02	2,16	1152,68	6358,18	1128,03	10,66
g13	875,13	882,57	861,92	1895,88	857,19	227154,13	904,86	225,22	925,81	0,35	875,71	6358,18	865,20	10,66
g14	1105,49	1196,02	1082,52	2814,61	1080,55	227154,13	1143,47	616,92	1155,19	0,52	1106,41	6358,18	1097,78	10,66
g15	1369,70	1784,80	1351,13	3835,09	1341,50	227154,13	1439,71	573,95	1461,49	0,84	1373,40	6358,18	1361,41	10,66
g16	1648,10	2611,98	1629,78	5619,15	1617,97	227154,13	1706,25	1461,58	1742,86	1,29	1682,88	6358,18	1635,58	10,66
g17	713,68	740,09	707,83	1594,68	707,76	227154,13	717,26	317,03	726,01	0,41	707,79	6358,18	711,74	10,66
g18	1017,65	1078,42	1000,27	3103,73	998,01	227154,13	1023,64	506,18	1077,53	0,59	1024,51	6358,18	1010,32	10,66
g19	1397,16	1583,32	1367,31	3665,85	1366,00	227154,13	1403,39	987,53	1444,51	0,81	1399,95	6358,18	1382,59	10,66
g20	1864,01	2166,08	1827,39	4998,61	1819,58	227154,13	1879,44	1886,75	1938,12	1,08	1821,15	6358,18	1850,92	10,66
Summe	13381,78	17047,70	13173,29	46582,15	13124,38	2725849,58	13692,41	15199,72	14072,78	11,36	13441,03	76298,15	13307,22	127,97

	LLYL		M		MM		MB		N		NBa		NBb	
g9	586,68	2550,08	589,94	4,14	586,87	18,49	583,39	266,66	580,60	1389,73	580,48	281,17	580,42	762,39
g10	748,89	4690,92	749,15	8,83	746,56	39,75	741,56	55,52	738,92	2245,78	738,73	521,15	738,46	1181,98
g11	924,70	8357,09	935,23	10,63	925,52	47,48	918,45	326,33	917,17	3434,04	914,75	710,98	914,72	1708,12
g12	1125,71	11837,19	1137,17	26,08	1114,31	117,45	1107,19	479,49	1108,48	6089,93	1106,33	916,94	1106,76	2602,85
g13	867,29	2300,89	875,14	10,49	865,19	47,34	859,11	296,12	857,19	1043,20	857,19	225,65	857,19	673,68
g14	1098,86	4487,94	1098,95	7,45	1089,21	33,54	1081,31	35,90	1080,55	1450,62	1080,55	274,01	1080,55	905,62
g15	1356,65	8322,96	1369,16	23,88	1355,28	107,37	1345,23	20,43	1340,24	1722,84	1341,23	456,68	1342,53	1368,11
g16	1642,90	10966,77	1651,14	36,71	1632,21	137,73	1622,69	592,32	2171,30	2882,44	1616,33	518,46	1620,85	1911,78
g17	712,26	2117,86	715,16	8,14	712,18	36,99	707,79	22,36	707,76	642,93	707,76	105,66	707,76	425,59
g18	1017,91	4120,70	1013,17	8,28	1006,31	37,40	998,73	111,49	995,39	1129,16	995,39	226,55	995,13	1071,20
g19	1384,93	7032,48	1384,18	9,94	1373,24	45,27	1366,86	17,25	1366,14	1802,53	1366,18	209,53	1365,97	1545,97
g20	1855,91	11065,35	1835,18	16,70	1831,17	75,22	1820,09	170,27	1820,54	2837,67	1819,99	354,60	1820,02	2064,14
Summe	13322,69	77850,22	13353,57	171,27	13238,05	744,01	13152,40	2394,14	13684,28	26670,89	13124,91	4801,39	13130,36	16221,43

	Pa		Pb		RDH		SWH		Ta		TV		Tb	
g9	596,89	121,13	586,18	58,20	588,62	134,42	593,35	877,66	585,43	127,13	587,09	5,45	587,32	-
g10	751,41	309,42	744,36	324,16	751,19	109,19	750,77	877,66	746,56	127,13	746,56	6,90	750,61	-
g11	939,74	664,02	922,40	201,97	938,00	605,16	932,81	877,66	923,17	127,13	932,68	8,52	933,00	-
g12	1152,88	261,12	1116,12	1292,81	1140,82	383,41	1138,32	877,66	1130,40	127,13	1137,18	10,32	1126,78	-
g13	877,71	129,42	860,55	95,27	870,28	544,70	872,45	877,66	865,01	127,13	881,04	5,38	867,10	-
g14	1089,93	288,19	1084,82	404,89	1096,56	161,47	1109,48	877,66	1086,07	127,13	1103,69	6,84	1085,01	-
g15	1371,61	934,53	1352,39	454,07	1366,34	148,67	1367,37	877,66	1353,91	127,13	1364,23	8,46	1356,81	-
g16	1650,94	1107,85	1634,27	935,00	1648,11	249,24	1651,31	877,66	1634,74	127,13	1656,66	10,25	1642,90	-
g17	717,09	49,57	707,79	45,62	710,33	427,89	710,42	877,66	708,74	127,13	720,44	5,13	710,45	-

Instanz	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit	Güte	Zeit
	Pa		Pb		RDH		SWH		Ta		TV		Tb	
g18	1018,74	332,69	1002,15	129,96	1006,48	266,91	1019,64	877,66	1006,90	127,13	1029,21	6,41	1015,20	-
g19	1385,60	628,07	1371,67	798,45	1377,87	704,61	1377,54	877,66	1371,01	127,13	1403,05	7,69	1377,00	-
g20	1846,55	1779,90	1830,98	256,51	1834,55	446,13	1850,59	877,66	1837,67	127,13	1875,17	8,97	1846,40	-
Summe	13399,09	6605,91	13213,68	4996,92	13329,12	4181,79	13374,05	10531,92	13249,61	1525,60	13437,00	90,31	13298,58	-

	VCGLR	
g9	579,71	2738,79
g10	736,26	5511,48
g11	912,84	6953,20
g12	1102,69	10412,58
g13	857,19	1354,29
g14	1080,55	2296,89
g15	1337,92	5782,76
g16	1612,50	9362,42
g17	707,76	1535,50
g18	995,13	2163,37
g19	1365,60	4317,73
g20	1818,32	4679,09
Summe	13106,47	57108,10

B.4 Lösungsgüte und Lösungszeit der Algorithmen

Die folgenden Tabellen enthalten die Lösungsgüte und Dauer der Metaheuristiken aus Kapitel 4.2. Die Zeit, die angegeben ist, ist dabei der Zeitpunkt, zu dem die beste Lösung (über alle fünf Durchläufe hinweg) gefunden wurde.

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	18,22	0,00	0,00
c2	835,26	852,20	850,75	24,04	2,03	1,85
c3	826,14	827,31	826,14	116,54	0,14	0,00
c4	1028,42	1031,60	1030,95	496,04	0,31	0,25
c5	1291,29	1306,55	1300,31	576,89	1,18	0,70
c11	1042,11	1042,11	1042,11	576,05	0,00	0,00
c12	819,56	819,56	819,56	92,34	0,00	0,00
Summe	6367,39	6403,96	6394,43	1900,11	0,57	0,42

Tabelle B.16: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1628,72	1627,06	14,64	0,64	0,54
T75b	1344,62	1353,42	1352,56	26,75	0,65	0,59
T75c	1291,01	1294,64	1294,06	21,85	0,28	0,24
T75d	1365,42	1365,58	1365,42	26,81	0,01	0,00
T100a	2041,34	2057,99	2055,17	113,97	0,82	0,68
T100b	1939,90	1944,21	1942,76	103,29	0,22	0,15
T100c	1406,20	1410,44	1409,22	76,59	0,30	0,21
T100d	1580,46	1586,02	1582,87	98,68	0,35	0,15
T150a	3055,23	3057,20	3055,92	598,85	0,06	0,02
T150b	2727,20	2739,06	2728,27	599,79	0,43	0,04
T150c	2341,84	2360,03	2358,70	496,41	0,78	0,72
T150d	2645,40	2664,17	2663,21	370,83	0,71	0,67
T385	24366,69	24473,63	24443,28	3403,85	0,44	0,31
Summe	47723,67	47935,11	47878,49	5952,32	0,44	0,32

Tabelle B.17: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,67	**2567,67**	608,01	-0,11	**-0,11**
C1_400	6765,03	6741,49	**6738,28**	3088,14	-0,35	**-0,40**
C1_600	13465,24	13504,82	13498,26	3601,38	0,29	0,25
C1_800	23999,35	23813,59	**23795,78**	3491,36	-0,77	**-0,85**
C1_1000	39811,15	40054,51	40035,36	3609,90	0,61	0,56
C2_200	1368,89	1484,26	1483,00	449,32	8,43	8,34
C2_400	2799,40	3133,14	3127,41	2500,31	11,92	11,72
C2_600	5441,98	6320,85	6316,76	2589,91	16,15	16,07
C2_800	8241,83	9706,21	9681,31	921,05	17,77	17,47
C2_1000	11792,66	14348,26	14308,41	3535,26	21,67	21,33
R1_200	2878,24	2875,80	**2874,99**	2460,79	-0,08	**-0,11**
R1_400	7192,03	7211,43	7204,41	3079,66	0,27	0,17
R1_600	15691,37	15743,99	**15679,00**	3540,13	0,34	**-0,08**
R1_800	27650,67	28537,14	28316,22	3506,73	3,21	2,41
R1_1000	42311,91	43937,22	43814,41	118,68	3,84	3,55
R2_200	1610,30	1612,90	1610,30	236,15	0,16	0,00
R2_400	3323,05	3334,50	**3312,88**	3569,20	0,34	**-0,31**
R2_600	6254,17	6255,03	**6229,18**	2505,22	0,01	**-0,40**
R2_800	10230,06	10302,17	10261,44	3247,03	0,70	0,31
R2_1000	15149,40	15202,74	15154,15	2668,32	0,35	0,03
RC1_200	2804,20	2817,98	2814,01	3146,70	0,49	0,35
RC1_400	7381,47	7282,81	**7271,66**	3533,62	-1,34	**-1,49**
RC1_600	14786,15	14973,28	14925,46	3431,91	1,27	0,94
RC1_800	26720,23	27220,96	27022,57	3511,26	1,87	1,13
RC1_1000	41583,54	42424,64	42152,89	778,08	2,02	1,37
RC2_200	1513,00	1514,72	1514,71	1684,84	0,11	0,11
RC2_400	3100,06	3116,23	3103,82	3340,19	0,52	0,12
RC2_600	5732,08	5759,37	5751,70	3423,17	0,48	0,34
RC2_800	9217,99	9390,42	9355,51	1441,00	1,87	1,49
RC2_1000	13631,76	13758,55	13680,40	3489,71	0,93	0,36
Summe	375017,70	384946,66	383601,97	77107,01	1,33	0,96

Tabelle B.18: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	4,66	0,00	0,00
c2	835,26	836,48	835,26	4,21	0,15	0,00
c3	826,14	826,14	826,14	92,70	0,00	0,00
c4	1028,42	1031,06	1028,42	239,66	0,26	0,00
c5	1291,29	1303,35	1300,99	491,40	0,93	0,75
c11	1042,11	1042,11	1042,11	133,40	0,00	0,00
c12	819,56	819,56	819,56	5,79	0,00	0,00
Summe	6367,39	6383,31	6377,10	971,82	0,25	0,15

Tabelle B.19: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1619,91	1618,36	7,23	0,10	0,00
T75b	1344,62	1344,83	1344,74	21,35	0,02	0,01
T75c	1291,01	1291,04	1291,01	5,56	0,00	0,00
T75d	1365,42	1365,42	1365,42	20,99	0,00	0,00
T100a	2041,34	2048,04	2045,85	115,53	0,33	0,22
T100b	1939,90	1941,00	1940,70	92,04	0,06	0,04
T100c	1406,20	1408,02	1406,20	100,48	0,13	0,00
T100d	1580,46	1581,44	1581,26	96,16	0,06	0,05
T150a	3055,23	3055,60	3055,23	178,11	0,01	0,00
T150b	2727,20	2732,96	2728,29	484,77	0,21	0,04
T150c	2341,84	2360,46	2358,66	560,20	0,79	0,72
T150d	2645,40	2655,06	2645,56	540,62	0,37	0,01
T385	24366,69	24579,21	24515,70	3346,88	0,87	0,61
Summe	47723,67	47982,97	47896,97	5569,92	0,54	0,36

Tabelle B.20: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,93	**2567,67**	601,67	-0,10	**-0,11**
C1_400	6765,03	6734,92	**6731,46**	3305,39	-0,45	**-0,50**
C1_600	13465,24	13459,51	**13450,30**	2781,63	-0,04	**-0,11**
C1_800	23999,35	23741,14	**23728,48**	3545,74	-1,08	**-1,13**
C1_1000	39811,15	39632,84	**39590,23**	592,69	-0,45	**-0,55**
C2_200	1368,89	1482,54	1482,43	1634,18	8,30	8,29
C2_400	2799,40	3128,54	3125,40	3148,54	11,76	11,65
C2_600	5441,98	6303,72	6298,84	3405,96	15,84	15,75
C2_800	8241,83	9685,15	9662,57	1382,58	17,51	17,24
C2_1000	11792,66	14390,75	14354,37	816,07	22,03	21,72
R1_200	2878,24	2875,19	**2870,49**	3544,82	-0,11	**-0,27**
R1_400	7192,03	7208,48	**7186,01**	3305,33	0,23	**-0,08**
R1_600	15691,37	15655,82	**15617,23**	3249,29	-0,23	**-0,47**
R1_800	27650,67	28002,24	27959,21	3620,47	1,27	1,12
R1_1000	42311,91	43039,94	42934,79	1495,86	1,72	1,47
R2_200	1610,30	1613,67	1610,30	491,60	0,21	0,00
R2_400	3323,05	3355,76	3347,29	3131,10	0,98	0,73
R2_600	6254,17	6247,81	**6218,15**	3579,62	-0,10	**-0,58**
R2_800	10230,06	10355,82	10281,49	3456,24	1,23	0,50
R2_1000	15149,40	15227,19	15217,47	2821,75	0,51	0,45
RC1_200	2804,20	2818,63	2809,68	1841,37	0,51	0,20
RC1_400	7381,47	7314,69	**7294,71**	3419,76	-0,90	**-1,18**
RC1_600	14786,15	14900,81	14873,83	3418,75	0,78	0,59
RC1_800	26720,23	26910,42	26819,72	3580,74	0,71	0,37
RC1_1000	41583,54	42054,43	42015,10	1213,53	1,13	1,04
RC2_200	1513,00	1513,34	1513,00	3171,02	0,02	0,00
RC2_400	3100,06	3116,33	3100,86	2832,81	0,52	0,03
RC2_600	5732,08	5735,91	**5723,67**	3611,55	0,07	**-0,15**
RC2_800	9217,99	9397,66	9379,29	3458,94	1,95	1,75
RC2_1000	13631,76	13836,89	13824,70	2366,64	1,50	1,42
Summe	375017,70	382308,11	381588,73	78825,65	0,56	0,37

Tabelle B.21: Lösungen der Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	20,02	0,00	0,00
c2	835,26	836,35	835,67	5,44	0,13	0,05
c3	826,14	827,14	826,14	40,04	0,12	0,00
c4	1028,42	1028,67	1028,42	274,76	0,02	0,00
c5	1291,29	1300,31	1297,54	359,34	0,70	0,48
c11	1042,11	1042,11	1042,11	278,54	0,00	0,00
c12	819,56	819,56	819,56	21,12	0,00	0,00
Summe	6367,39	6378,76	6374,05	999,25	0,18	0,10

Tabelle B.22: Lösungen der Multistart-TS_{GPU} auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1618,39	1618,36	13,95	0,00	0,00
T75b	1344,62	1344,92	1344,67	0,72	0,02	0,00
T75c	1291,01	1291,01	1291,01	3,58	0,00	0,00
T75d	1365,42	1365,42	1365,42	12,90	0,00	0,00
T100a	2041,34	2049,08	2048,46	54,67	0,38	0,35
T100b	1939,90	1941,25	1941,04	101,15	0,07	0,06
T100c	1406,20	1409,91	1406,30	18,94	0,26	0,01
T100d	1580,46	1581,21	1580,46	57,62	0,05	0,00
T150a	3055,23	3055,29	3055,23	147,99	0,00	0,00
T150b	2727,20	2730,20	2728,12	183,71	0,11	0,03
T150c	2341,84	2361,70	2361,57	260,28	0,85	0,84
T150d	2645,40	2651,21	2645,39	93,15	0,22	0,00
T385	24366,69	24552,95	24518,29	1667,44	0,76	0,62
Summe	47723,67	47952,54	47904,31	2616,10	0,48	0,38

Tabelle B.23: Lösungen der Multistart-TS_{GPU} auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,95	2567,67	1319,56	-0,10	-0,11
C1_400	6765,03	6729,16	6721,28	3529,76	-0,53	-0,65
C1_600	13465,24	13457,01	13454,98	2378,29	-0,06	-0,08
C1_800	23999,35	23746,08	23735,78	3028,98	-1,06	-1,10
C1_1000	39811,15	39616,22	39567,37	3551,96	-0,49	-0,61
C2_200	1368,89	1482,43	1482,43	1672,94	8,29	8,29
C2_400	2799,40	3142,80	3130,99	1486,26	12,27	11,85
C2_600	5441,98	6293,24	6284,85	3398,59	15,64	15,49
C2_800	8241,83	9654,34	9626,61	3371,16	17,14	16,80
C2_1000	11792,66	14332,11	14293,59	3601,93	21,53	21,21
R1_200	2878,24	2873,66	2869,81	2891,35	-0,16	-0,29
R1_400	7192,03	7191,33	7176,58	2522,94	-0,01	-0,21
R1_600	15691,37	15563,31	15517,54	2940,61	-0,82	-1,11
R1_800	27650,67	27798,95	27772,71	3589,47	0,54	0,44
R1_1000	42311,91	42889,54	42765,54	3591,00	1,37	1,07
R2_200	1610,30	1614,06	1610,30	93,78	0,23	0,00
R2_400	3323,05	3334,77	3318,24	2284,48	0,35	-0,14
R2_600	6254,17	6325,09	6282,18	3382,39	1,13	0,45
R2_800	10230,06	10461,23	10337,10	3293,71	2,26	1,05
R2_1000	15149,40	15696,17	15530,79	3609,01	3,61	2,52
RC1_200	2804,20	2819,19	2804,95	2176,28	0,53	0,03
RC1_400	7381,47	7291,34	7278,29	838,86	-1,22	-1,40
RC1_600	14786,15	14899,98	14871,57	3357,83	0,77	0,58
RC1_800	26720,23	26879,67	26770,90	3598,76	0,60	0,19
RC1_1000	41583,54	42000,25	41936,29	3600,46	1,00	0,85
RC2_200	1513,00	1513,34	1513,00	2214,63	0,02	0,00
RC2_400	3100,06	3107,12	3090,19	1658,26	0,23	-0,32
RC2_600	5732,08	5843,75	5766,66	3558,50	1,95	0,60
RC2_800	9217,99	9517,64	9433,13	3539,29	3,25	2,33
RC2_1000	13631,76	14294,52	14240,32	3594,11	4,86	4,46
Summe	375017,70	382936,24	381751,63	83675,13	0,77	0,45

Tabelle B.24: Lösungen der Multistart-TS_{GPU} auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	7,41	0,00	0,00
c2	835,26	835,26	835,26	25,88	0,00	0,00
c3	826,14	826,14	826,14	118,48	0,00	0,00
c4	1028,42	1029,91	1029,79	567,30	0,14	0,13
c5	1291,29	1298,11	1292,48	573,10	0,53	0,09
c11	1042,11	1042,12	1042,11	545,20	0,00	0,00
c12	819,56	819,56	819,56	30,65	0,00	0,00
Summe	6367,39	6375,71	6369,94	1868,01	0,13	0,04

Tabelle B.25: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1619,83	1618,36	26,06	0,09	0,00
T75b	1344,62	1344,63	1344,62	10,47	0,00	0,00
T75c	1291,01	1291,01	1291,01	16,97	0,00	0,00
T75d	1365,42	1365,42	1365,42	16,77	0,00	0,00
T100a	2041,34	2049,06	2047,90	39,36	0,38	0,32
T100b	1939,90	1940,67	1940,61	52,38	0,04	0,04
T100c	1406,20	1408,02	1406,20	35,23	0,13	0,00
T100d	1580,46	1584,91	1581,26	50,12	0,28	0,05
T150a	3055,23	3055,58	3055,23	183,72	0,01	0,00
T150b	2727,20	2733,61	2727,88	434,74	0,23	0,02
T150c	2341,84	2362,05	2361,44	549,18	0,86	0,84
T150d	2645,40	2659,83	2649,26	252,57	0,55	0,15
T385	24366,69	24549,74	24492,99	3280,98	0,75	0,52
Summe	47723,67	47964,35	47882,18	4948,54	0,50	0,33

Tabelle B.26: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,77	**2567,67**	2963,13	-0,11	**-0,11**
C1_400	6765,03	6729,55	**6725,28**	3307,97	-0,52	**-0,59**
C1_600	13465,24	13449,72	**13440,16**	2902,75	-0,12	**-0,19**
C1_800	23999,35	23728,49	**23705,29**	3369,28	-1,13	**-1,23**
C1_1000	39811,15	39508,89	**39441,30**	3625,88	-0,76	**-0,93**
C2_200	1368,89	1483,31	1482,43	1794,53	8,36	8,29
C2_400	2799,40	3133,07	3124,63	2253,96	11,92	11,62
C2_600	5441,98	6319,53	6304,96	3603,76	16,13	15,86
C2_800	8241,83	9654,78	9612,74	3334,62	17,14	16,63
C2_1000	11792,66	14345,04	14317,84	3360,71	21,64	21,41
R1_200	2878,24	2876,54	**2869,66**	1692,17	-0,06	**-0,30**
R1_400	7192,03	7195,83	**7187,14**	2719,66	0,05	**-0,07**
R1_600	15691,37	15608,94	**15573,38**	3603,93	-0,53	**-0,75**
R1_800	27650,67	27858,53	27775,81	3542,53	0,75	0,45
R1_1000	42311,91	42883,41	42832,43	3627,92	1,35	1,23
R2_200	1610,30	1618,40	1611,74	3190,80	0,50	0,09
R2_400	3323,05	3361,43	3343,06	3559,13	1,15	0,60
R2_600	6254,17	6332,39	6306,63	3605,52	1,25	0,84
R2_800	10230,06	10465,26	10388,83	3578,88	2,30	1,55
R2_1000	15149,40	15361,71	15249,71	3476,73	1,40	0,66
RC1_200	2804,20	2819,47	**2800,78**	3461,15	0,54	**-0,12**
RC1_400	7381,47	7302,15	**7273,88**	3054,27	-1,07	**-1,46**
RC1_600	14786,15	14860,33	14833,25	3592,62	0,50	0,32
RC1_800	26720,23	26829,50	26760,54	3514,86	0,41	0,15
RC1_1000	41583,54	41897,40	41717,23	3600,66	0,75	0,32
RC2_200	1513,00	1514,37	1513,00	2306,82	0,09	0,00
RC2_400	3100,06	3120,64	3106,24	3586,98	0,66	0,20
RC2_600	5732,08	5794,29	5739,38	3598,16	1,09	0,13
RC2_800	9217,99	9499,52	9406,53	3318,89	3,05	2,05
RC2_1000	13631,76	13900,81	13720,54	3631,84	1,97	0,65
Summe	375017,70	382021,05	380732,08	96780,11	0,50	0,15

Tabelle B.27: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	25,04	0,00	0,00
c2	835,26	844,93	842,37	16,49	1,16	0,85
c3	826,14	826,14	826,14	48,14	0,00	0,00
c4	1028,42	1030,34	1030,03	410,75	0,19	0,16
c5	1291,29	1299,64	1292,44	550,99	0,65	0,09
c11	1042,11	1042,11	1042,11	198,09	0,00	0,00
c12	819,56	819,56	819,56	88,98	0,00	0,00
Summe	6367,39	6387,33	6377,26	1338,46	0,31	0,16

Tabelle B.28: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1625,00	1623,40	28,16	0,41	0,31
T75b	1344,62	1347,98	1346,62	27,87	0,25	0,15
T75c	1291,01	1292,46	1291,01	22,45	0,11	0,00
T75d	1365,42	1365,50	1365,42	22,91	0,01	0,00
T100a	2041,34	2072,50	2066,52	115,58	1,53	1,23
T100b	1939,90	1944,50	1943,22	100,14	0,24	0,17
T100c	1406,20	1407,49	1406,85	114,39	0,09	0,05
T100d	1580,46	1583,72	1582,75	114,38	0,21	0,14
T150a	3055,23	3062,60	3059,79	26,15	0,24	0,15
T150b	2727,20	2749,64	2744,71	24,83	0,82	0,64
T150c	2341,84	2370,12	2367,91	28,74	1,21	1,11
T150d	2645,40	2680,15	2676,10	25,05	1,31	1,16
T385	24366,69	24479,81	24472,31	3509,94	0,46	0,43
Summe	47723,67	47981,47	47946,60	4160,59	0,54	0,47

Tabelle B.29: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,67	**2567,67**	968,74	-0,11	**-0,11**
C1_400	6765,03	6736,55	**6730,88**	2473,78	-0,42	**-0,50**
C1_600	13465,24	13485,24	13476,80	3607,25	0,15	0,09
C1_800	23999,35	23780,95	**23767,77**	3618,28	-0,91	**-0,96**
C1_1000	39811,15	39803,14	**39662,48**	3397,72	-0,02	**-0,37**
C2_200	1368,89	1484,14	1482,43	595,04	8,42	8,29
C2_400	2799,40	3140,34	3125,99	3567,11	12,18	11,67
C2_600	5441,98	6311,24	6303,97	3484,95	15,97	15,84
C2_800	8241,83	9623,88	9607,58	3476,69	16,77	16,57
C2_1000	11792,66	14333,56	14289,51	3275,91	21,55	21,17
R1_200	2878,24	2880,00	**2871,26**	2050,45	0,06	**-0,24**
R1_400	7192,03	7207,12	7193,66	3552,13	0,21	0,02
R1_600	15691,37	15699,14	**15664,68**	3596,81	0,05	**-0,17**
R1_800	27650,67	28073,06	27972,24	3442,70	1,53	1,16
R1_1000	42311,91	43405,83	43202,86	3537,41	2,59	2,11
R2_200	1610,30	1616,48	1616,48	220,88	0,38	0,38
R2_400	3323,05	3344,32	**3319,54**	2363,20	0,64	**-0,11**
R2_600	6254,17	6255,79	**6233,44**	3537,68	0,03	**-0,33**
R2_800	10230,06	10293,02	10232,92	3527,18	0,62	0,03
R2_1000	15149,40	15178,92	**15090,29**	3559,28	0,19	**-0,39**
RC1_200	2804,20	2815,15	2803,36	3171,20	0,39	-0,03
RC1_400	7381,47	7307,95	**7267,21**	3454,84	-1,00	**-1,55**
RC1_600	14786,15	14879,35	14836,39	3462,69	0,63	0,34
RC1_800	26720,23	26999,97	26889,81	3400,60	1,05	0,63
RC1_1000	41583,54	41980,84	41848,51	3473,16	0,96	0,64
RC2_200	1513,00	1514,37	1513,00	343,04	0,09	0,00
RC2_400	3100,06	3115,11	3106,20	3407,63	0,49	0,20
RC2_600	5732,08	5776,56	5763,01	3388,86	0,78	0,54
RC2_800	9217,99	9408,13	9329,75	3616,86	2,06	1,21
RC2_1000	13631,76	13716,37	13669,40	3483,91	0,62	0,28
Summe	375017,70	382734,21	381439,07	89055,98	0,71	0,36

Tabelle B.30: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit adaptivem Abkühlungsplan auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	2,67	0,00	0,00
c2	835,26	835,27	835,26	24,39	0,00	0,00
c3	826,14	826,64	826,14	65,55	0,06	0,00
c4	1028,42	1030,34	1028,42	152,73	0,19	0,00
c5	1291,29	1300,86	1297,59	530,17	0,74	0,49
c11	1042,11	1042,11	1042,11	262,50	0,00	0,00
c12	819,56	819,56	819,56	5,32	0,00	0,00
Summe	6367,39	6379,40	6373,69	1043,32	0,19	0,10

Tabelle B.31: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1618,56	1618,36	17,63	0,01	0,00
T75b	1344,62	1344,64	1344,62	23,42	0,00	0,00
T75c	1291,01	1291,01	1291,01	13,81	0,00	0,00
T75d	1365,42	1365,42	1365,42	14,63	0,00	0,00
T100a	2041,34	2047,41	2041,34	100,33	0,30	0,00
T100b	1939,90	1940,63	1940,61	43,12	0,04	0,04
T100c	1406,20	1406,20	1406,20	24,12	0,00	0,00
T100d	1580,46	1582,89	1580,46	104,83	0,15	0,00
T150a	3055,23	3055,23	3055,23	362,73	0,00	0,00
T150b	2727,20	2732,51	2727,88	479,99	0,19	0,02
T150c	2341,84	2361,97	2361,44	449,63	0,86	0,84
T150d	2645,40	2653,63	2645,39	375,99	0,31	0,00
T385	24366,69	24527,31	24505,20	3599,44	0,66	0,57
Summe	47723,67	47927,41	47883,15	5609,67	0,43	0,33

Tabelle B.32: Lösungen des GA mit Multistart-$VNS \times SimA_{GPU}$ mit COSA auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,84	**2567,67**	739,06	-0,10	**-0,11**
C1_400	6765,03	6730,62	**6730,04**	2696,52	-0,51	**-0,52**
C1_600	13465,24	13442,06	**13436,00**	3228,73	-0,17	**-0,22**
C1_800	23999,35	23731,57	**23713,51**	2259,85	-1,12	**-1,19**
C1_1000	39811,15	39429,41	**39379,69**	3601,88	-0,96	**-1,08**
C2_200	1368,89	1484,91	1482,43	2696,88	8,48	8,29
C2_400	2799,40	3129,44	3124,22	1004,96	11,79	11,60
C2_600	5441,98	6286,31	6269,95	3538,73	15,52	15,21
C2_800	8241,83	9670,79	9601,51	3296,07	17,34	16,50
C2_1000	11792,66	14303,16	14258,22	3414,80	21,29	20,91
R1_200	2878,24	2878,34	**2871,26**	1909,12	0,00	**-0,24**
R1_400	7192,03	7208,94	**7184,40**	3388,34	0,24	**-0,11**
R1_600	15691,37	15591,66	**15562,22**	2981,84	-0,64	**-0,82**
R1_800	27650,67	27916,47	27880,56	3259,72	0,96	0,83
R1_1000	42311,91	42811,24	42737,05	3609,89	1,18	1,00
R2_200	1610,30	1614,60	1610,30	578,10	0,27	0,00
R2_400	3323,05	3348,69	3340,40	1798,36	0,77	0,52
R2_600	6254,17	6225,03	**6188,39**	3507,79	-0,47	**-1,05**
R2_800	10230,06	10404,26	10313,08	3501,52	1,70	0,81
R2_1000	15149,40	15161,10	**15112,32**	3305,08	0,08	**-0,24**
RC1_200	2804,20	2819,25	2813,63	2336,24	0,54	0,34
RC1_400	7381,47	7322,64	**7299,03**	3505,41	-0,80	**-1,12**
RC1_600	14786,15	14849,96	14838,30	3540,47	0,43	0,35
RC1_800	26720,23	26863,21	26821,16	3450,44	0,54	0,38
RC1_1000	41583,54	41882,66	41780,51	3646,21	0,72	0,47
RC2_200	1513,00	1514,02	1513,00	807,00	0,07	0,00
RC2_400	3100,06	3110,68	**3094,14**	3449,62	0,34	**-0,19**
RC2_600	5732,08	5748,69	**5715,92**	3581,03	0,29	**-0,28**
RC2_800	9217,99	9421,52	9369,43	3621,78	2,21	1,64
RC2_1000	13631,76	13758,49	13704,25	3315,64	0,93	0,53
Summe	375017,70	381227,58	380312,60	85571,11	0,28	0,06

Tabelle B.33: Lösungen des GA mit Mulitstart-$VNS \times SimA_{GPU}$ mit COSA auf den Gehring-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
c1	524,61	524,61	524,61	17,41	0,00	0,00
c2	835,26	836,01	835,26	30,15	0,09	0,00
c3	826,14	826,89	826,14	41,08	0,09	0,00
c4	1028,42	1028,74	1028,42	485,63	0,03	0,00
c5	1291,29	1298,09	1293,53	582,28	0,53	0,17
c11	1042,11	1042,11	1042,11	174,31	0,00	0,00
c12	819,56	819,56	819,56	15,25	0,00	0,00
Summe	6367,39	6376,02	6369,63	1346,11	0,14	0,04

Tabelle B.34: Lösungen des GA mit Multistart-TS_{GPU} auf den Christofides-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
T75a	1618,36	1618,36	1618,36	2,39	0,00	0,00
T75b	1344,62	1344,78	1344,74	2,08	0,01	0,01
T75c	1291,01	1291,01	1291,01	7,61	0,00	0,00
T75d	1365,42	1365,42	1365,42	5,14	0,00	0,00
T100a	2041,34	2048,64	2048,19	63,30	0,36	0,34
T100b	1939,90	1941,02	1940,70	55,38	0,06	0,04
T100c	1406,20	1408,12	1406,24	95,35	0,14	0,00
T100d	1580,46	1581,28	1580,57	110,68	0,05	0,01
T150a	3055,23	3055,23	3055,23	220,90	0,00	0,00
T150b	2727,20	2728,34	2727,99	344,47	0,04	0,03
T150c	2341,84	2361,01	2358,71	591,45	0,82	0,72
T150d	2645,40	2645,39	2645,39	237,74	0,00	0,00
T385	24366,69	24501,65	24480,23	3486,29	0,55	0,47
Summe	47723,67	47890,23	47862,76	5222,79	0,35	0,29

Tabelle B.35: Lösungen des GA mit Multistart-TS_{GPU} auf den Taillard-Instanzen

Instanz	Beste bekannte Lösung	⌀Lösungsgüte	Beste Lösung	Zeit [s]	⌀Abweichung [%]	Abweichung beste Lösung [%]
C1_200	2570,49	2567,86	**2567,67**	1820,06	-0,10	**-0,11**
C1_400	6765,03	6726,14	**6721,10**	2519,70	-0,57	**-0,65**
C1_600	13465,24	13448,62	**13437,92**	2846,08	-0,12	**-0,20**
C1_800	23999,35	23728,80	**23712,53**	3524,81	-1,13	**-1,20**
C1_1000	39811,15	39467,20	**39364,68**	3557,73	-0,86	**-1,12**
C2_200	1368,89	1482,43	1482,43	1317,59	8,29	8,29
C2_400	2799,40	3136,73	3123,99	465,83	12,05	11,59
C2_600	5441,98	6287,39	6280,19	3386,17	15,53	15,40
C2_800	8241,83	9677,24	9651,14	3308,55	17,42	17,10
C2_1000	11792,66	14336,56	14287,64	3571,32	21,57	21,16
R1_200	2878,24	2873,68	**2871,44**	3264,89	-0,16	**-0,24**
R1_400	7192,03	7182,96	**7176,31**	2896,71	-0,13	**-0,22**
R1_600	15691,37	15577,47	**15525,05**	3486,05	-0,73	**-1,06**
R1_800	27650,67	27758,06	27682,63	3505,58	0,39	0,12
R1_1000	42311,91	42878,69	42811,79	3589,97	1,34	1,18
R2_200	1610,30	1611,53	1610,30	3260,22	0,08	0,00
R2_400	3323,05	3336,29	**3312,87**	1874,25	0,40	**-0,31**
R2_600	6254,17	6317,45	6267,77	2211,38	1,01	0,22
R2_800	10230,06	10431,18	10324,43	3552,67	1,97	0,92
R2_1000	15149,40	15540,89	15344,56	3602,27	2,58	1,29
RC1_200	2804,20	2813,10	**2800,78**	3221,36	0,32	**-0,12**
RC1_400	7381,47	7292,61	**7269,95**	3520,34	-1,20	**-1,51**
RC1_600	14786,15	14860,45	14801,89	3568,88	0,50	0,11
RC1_800	26720,23	26740,24	**26655,82**	3563,37	0,07	**-0,24**
RC1_1000	41583,54	41934,95	41910,91	3527,01	0,85	0,79
RC2_200	1513,00	1513,00	1513,00	3549,50	0,00	0,00
RC2_400	3100,06	3114,44	**3097,45**	3321,79	0,46	**-0,08**
RC2_600	5732,08	5815,84	5766,39	3450,22	1,46	0,60
RC2_800	9217,99	9448,10	9372,70	3455,29	2,50	1,68
RC2_1000	13631,76	14101,38	14013,79	3616,96	3,45	2,80
Summe	375017,70	382001,28	380759,10	92356,56	0,49	0,16

Tabelle B.36: Lösungen des GA mit Multistart-TS_{GPU} auf den Gehring-Instanzen

Instanz	Beste bekannte	VC	TC	VG	VGA	VGC	TG	GVG	GVGA	GVGC	GTG
c1	524,61	524,61	524,61	524,61	524,61	524,61	524,61	524,61	524,61	524,61	524,61
c2	835,26	835,32	837,41	835,26	850,75	835,26	835,67	835,26	842,37	835,26	835,26
c3	826,14	826,21	827,39	826,14	826,14	826,14	826,14	826,14	826,14	826,14	826,14
c4	1028,42	1029,87	1028,78	1028,78	1030,95	1028,42	1028,42	1029,79	1030,03	1028,42	1028,42
c5	1291,29	1303,94	1302,54	1293,13	1300,31	1300,99	1297,54	1292,48	1292,44	1297,59	1293,53
c11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11	1042,11
c12	819,56	819,56	819,56	819,56	819,56	819,56	819,56	819,56	819,56	819,56	819,56
Summe	6367,39	6381,62	6382,40	6369,59	6394,43	6377,10	6374,05	6369,94	6377,26	6373,69	6369,63
Zeit		1253,03	1063,20	1694,05	1900,11	971,82	999,25	1868,01	1338,46	1043,32	1346,11
Abweichung	0,00	0,22	0,24	0,03	0,42	0,15	0,10	0,04	0,16	0,10	0,04

Tabelle B.37: Beste Lösungen der implementierten Metaheuristiken für die Christofides-Instanzen

Instanz	Beste bekannte	VC	TC	VG	VGA	VGC	TG	GVG	GVGA	GVGC	GTG
T75a	1618,36	1618,36	1619,69	1620,81	1627,06	1618,36	1618,36	1618,36	1623,40	1618,36	1618,36
T75b	1344,62	1344,64	1345,07	1344,62	1352,56	1344,74	1344,67	1344,62	1346,62	1344,62	1344,74
T75c	1291,01	1291,01	1291,01	1291,01	1294,06	1291,01	1291,01	1291,01	1291,01	1291,01	1291,01
T75d	1365,42	1365,42	1368,01	1365,42	1365,42	1365,42	1365,42	1365,42	1365,42	1365,42	1365,42
T100a	2041,34	2056,46	2048,40	2045,80	2055,17	2045,85	2048,46	2047,90	2066,52	2041,34	2048,19
T100b	1939,90	1941,38	1940,70	1940,72	1942,76	1940,70	1941,04	1940,61	1943,22	1940,61	1940,70
T100c	1406,20	1406,26	1415,28	1406,20	1409,22	1406,20	1406,30	1406,20	1406,85	1406,20	1406,24
T100d	1580,46	1585,07	1598,41	1581,26	1582,87	1581,26	1580,46	1581,26	1582,75	1580,46	1580,57
T150a	3055,23	3057,21	3108,29	3055,23	3055,92	3055,23	3055,23	3055,23	3059,79	3055,23	3055,23
T150b	2727,20	2732,12	2737,34	2728,17	2728,27	2728,29	2728,12	2727,88	2744,71	2727,88	2727,99
T150c	2341,84	2361,62	2364,22	2361,44	2358,70	2358,66	2361,57	2361,44	2367,91	2361,44	2358,71
T150d	2645,40	2662,82	2661,12	2659,16	2663,21	2645,56	2645,39	2649,26	2676,10	2645,39	2645,39
T385	24366,69	24674,85	24677,67	24555,36	24443,28	24515,70	24518,29	24492,99	24472,31	24505,20	24480,23
Summe	47723,67	48097,22	48175,20	47955,19	47878,49	47896,97	47904,31	47882,18	47946,60	47883,15	47862,76
Zeit		4545,67	1949,52	5942,71	5952,32	5569,92	2616,10	4948,54	4160,59	5609,67	5222,79
Abweichung	0,00	0,78	0,95	0,49	0,32	0,36	0,38	0,33	0,47	0,33	0,29

Tabelle B.38: Beste Lösungen der implementierten Metaheuristiken für die Taillard-Instanzen

C Software

Die Software, die im Rahmen dieser Arbeit entwickelt wurde, kann per Mail an sand@wiwi.uni-kl.de angefordert werden.

Literaturverzeichnis

[Alabas-Uslu und Dengiz 2011] Alabas-Uslu, C.; Dengiz, B.: A self-adaptive local search algorithm for the classical vehicle routing problem. In: *Expert Systems with Applications* 38 (2011), Nr. 7, S. 8990–8998

[Alba und Dorronsoro 2004] Alba, E.; Dorronsoro, B.: Solving the vehicle routing problem by using cellular genetic algorithms. In: Gottlieb, J. (Hrsg.); Raidl, G. (Hrsg.): *Evolutionary Computation in Combinatorial Optimization*. Berlin/Heidelberg: Springer, 2004, S. 11–20

[Alba und Dorronsoro 2006] Alba, E.; Dorronsoro, B.: Computing nine new best-so-far solutions for capacitated vrp with a cellular genetic algorithm. In: *Information Processing Letters* 98 (2006), S. 225–230

[Alba und Dorronsoro 2008] Alba, E.; Dorronsoro, B.: *Cellular Genetic Algorithms*. Berlin: Springer, 2008

[Alba et al. 2012] Alba, E.; Luque, G.; Nesmachnow, S.: Parallel metaheuristics: Recent advances and new trends. In: *International Transactions in Operational Research* (2012), S. 1–48

[Amazon.com 2012a] Amazon.com: *Amazon EC2 – Preise*. 2012. – URL `http://aws.amazon.com/de/ec2/pricing/`. – Zugriffsdatum: 14.12.2012

[Amazon.com 2012b] Amazon.com: *Amazon EC2-Instancetypen*. 2012. – URL `http://aws.amazon.com/de/ec2/instance-types/`. – Zugriffsdatum: 14.12.2012

[Amdahl 1967] Amdahl, G. M.: Validity of the single processor approach to achieving large scale computing capabilities. In: *American Federation of Information Processing Societies Spring Joint Computer Conference*, 1967, S. 483–485

[Andermahr 2010] Andermahr, W.: *Test: Nvidia GeForce GTX 580 (2/29): Schneller, leiser, etwas sparsamer*. 2010. – URL `http://www.computerbase.de/artikel/grafikkarten/2010/test-nvidia-geforce-gtx-580/2/`. – Zugriffsdatum: 14.12.2012

[Anderson et al. 2011] Anderson, M.; Catanzaro, B.; Chong, J.; Gonina, E.; Keutzer, K.; Lai, C.-Y.; Murphy, M.; Sheffield, D.; Su, B.-Y.; Sundaram, N.: Considerations when evaluating microprocessor platforms. In: *Proceedings of the 3rd USENIX conference on Hot topic in parallelism*, USENIX Association, 2011

[Arnold et al. 2008] Arnold, D.; Isermann, H.; Kuhn, A.; Tempelmeier, H.; Furmans, K.: *Handbuch Logistik*. 3. Auflage. Berlin: Springer, 2008

[Baldacci et al. 2010] Baldacci, R.; Vigo, D.; Toth, P.: Exact solution of the capacitated vehicle routing problem. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc, 2010

[Barbucha 2011] Barbucha, D.: Solving the capacitated vehicle routing problem by a team of parallel heterogeneous cooperating agents. In: Jedrzejowicz, P. (Hrsg.); Nguyen, N. (Hrsg.); Hoang, K. (Hrsg.): *Computational Collective Intelligence. Technologies and Applications*. Berlin/Heidelberg: Springer, 2011, S. 332–341

[Barr et al. 1995] Barr, R.; Golden, B.; Kelly, J.; Resende, M.; Stewart, W.: Designing and reporting on computational experiments with heuristic methods. In: *Journal of Heuristics* 1 (1995), S. 9–32

[Ben-Ameur 2004] Ben-Ameur, W.: Computing the initial temperature of simulated annealing. In: *Computational Optimization and Applications* 29 (2004), Nr. 3, S. 369–385

[Bengel et al. 2008] Bengel, G.; Baun, C.; Kunze, M.; Stucky, K.-U.: *Masterkurs Parallele und Verteilte Systeme: Grundlagen und Programmierung von Multicoreprozessoren, Multiprozessoren, Cluster und Grid*. Wiesbaden: Vieweg + Teubner, 2008

[Berger und Barkaoui 2003] Berger, J.; Barkaoui, M.: A new hybrid genetic algorithm for the capacitated vehicle routing problem. In: *Journal of the Operational Research Society* 54 (2003), Nr. 12, S. 1254–1262

[Blake et al. 2009] Blake, G.; Dreslinski, R. G.; Mudge, T.: A survey of multicore processors. In: *IEEE Signal Processing Magazine* 26 (2009), Nr. 6, S. 26–37

[Blum und Roli 2003] Blum, C.; Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. In: *ACM Computing Surveys* 35 (2003), Nr. 3, S. 268–308

[Bolz et al. 2003] Bolz, J.; Farmer, I.; Grinspun, E.; Schröder, P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In: *ACM Transactions on Graphics* 22 (2003), Nr. 3, S. 917–924

[Borgo und Brodlie 2009] Borgo, R.; Brodlie, K.: *State of the art report on GPU*. 2009. – URL `http://www.comp.leeds.ac.uk/viznet/reports/GPU_report/GPU_final.pdf`. – Zugriffsdatum: 28.12.2012

[Bräysy und Gendreau 2005] Bräysy, O.; Gendreau, M.: Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. In: *Transportation Science* 39 (2005), Nr. 1, S. 104–118

[van Breedam 2002] Breedam, A. van: A parametric analysis of heuristics for the vehicle routing problem with side-constraints. In: *European Journal of Operational Research* 137 (2002), Nr. 2, S. 348–370

[Brodtkorb et al. 2010] Brodtkorb, A. R.; Dyken, C.; Hagen, T. R.; Hjelmervik, J. M.; Storaasli, O. O.: State-of-the-art in heterogeneous computing. In: *Scientific Programming* 18 (2010), Nr. 1, S. 1–33

[Buchty et al. 2009] Buchty, R.; Heuveline, V.; Karl, W.; Weiss, J.-P.: *A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators.* 2009. – URL `http://www.emcl.kit.edu/preprints/emcl-preprint-2009-02.pdf`. – Zugriffsdatum: 28.12.2012

[Bundesverband Güterkraftverkehr Logistik und Entsorgung (BGL) e.V. 2012] Bundesverband Güterkraftverkehr Logistik und Entsorgung (BGL) e.V.: *Dieselpreis-Information (Großverbraucher) – Vorabfassung für November 2012.* 2012. – URL `http://www.bgl-ev.de/images/downloads/initiativen/dieselpreisinformation.pdf?ms=629`. – Zugriffsdatum: 18.12.2012

[Černý 1985] Černý, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. In: *Journal of Optimization Theory and Applications* 45 (1985), Nr. 1, S. 41–51

[Che et al. 2008] Che, S.; Boyer, M.; Meng, J.; Tarjan, D.; Sheaffer, J. W.; Skadron, K.: A performance study of general-purpose applications on graphics processors using CUDA. In: *Journal of Parallel and Distributed Computing* 68 (2008), Nr. 10, S. 1370–1380

[Chen et al. 2010] Chen, P.; Huang, H.-K.; Dong, X.-Y.: Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. In: *Expert Systems with Applications* 37 (2010), Nr. 2, S. 1620–1627

[Christofides und Eilon 1969] Christofides, N.; Eilon, S.: An algorithm for the vehicle-dispatching problem. In: *Operational Research Quarterly* 20 (1969), Nr. 3, S. 309–318

[Christofides et al. 1979] Christofides, N.; Mingozzi, A.; Toth, P.: The vehicle routing problem. In: Christofides, N. (Hrsg.); Mingozzi, A. (Hrsg.); Toth, P. (Hrsg.); Sandi, C. (Hrsg.): *Combinatorial Optimization.* New York: John Wiley, 1979, S. 315–338

[Clarke und Wright 1964] Clarke, G.; Wright, J. W.: Scheduling of vehicles from a central depot to a number of delivery points. In: *Operations Research* 12 (1964), Nr. 4, S. 568–581

[Coelho et al. 2012] Coelho, I. M.; Ochi, L. S.; Munhoz, P. L. A.; Souza, M. J. F.; Farias, R.; Bentes, C.: the single vehicle routing problem with deliveries and selective pickups in a CPU-GPU heterogeneous environment. In: *International Conference on High Performance Computing and Communications*, 2012, S. 1606–1611

[Cohn und Fielding 1999] Cohn, H.; Fielding, M.: Simulated annealing: Searching for an optimal temperature schedule. In: *SIAM Journal on Optimization* 9 (1999), Nr. 3, S. 779–802

[Cordeau et al. 2005] Cordeau, J.-F.; Gendreau, M.; Hertz, A.; Laporte, G.; Sormany, J.-S.: New heuristics for the vehicle routing problem. In: Langevin, A. (Hrsg.); Riopel, D. (Hrsg.): *Logistics systems: Design and optimization.* New York: Springer, 2005, S. 279–297

[Cordeau et al. 1997] Cordeau, J.-F.; Gendreau, M.; Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. In: *Networks* 30 (1997), Nr. 2, S. 105–119

[Cordeau und Maischberger 2012] Cordeau, J.-F.; Maischberger, M.: A parallel iterated tabu search heuristic for vehicle routing problems. In: *Computers & Operations Research* 39 (2012), Nr. 9, S. 2033–2050

[Crainic 2008] Crainic, T. G.: Parallel solution methods for vehicle routing problems. In: Golden, B. L. (Hrsg.); Raghavan, S. (Hrsg.); Wasil, E. A. (Hrsg.): *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York/London: Springer, 2008, S. 171–198

[Crainic und Toulouse 2010] Crainic, T. G.; Toulouse, M.: Parallel meta-heuristics. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics*. New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 497–541

[Dantzig und Ramser 1959] Dantzig, G. B.; Ramser, J. H.: The truck dispatching problem. In: *Management Science* 6 (1959), Nr. 1, S. 80–91

[Doerner et al. 2004] Doerner, K.; Hartl, R.; Kiechle, G.; Lucka, M.; Reimann, M.: Parallel ant systems for the capacitated vehicle routing problem. In: Gottlieb, J. (Hrsg.); Raidl, G. (Hrsg.): *Evolutionary Computation in Combinatorial Optimization*. Berlin/Heidelberg: Springer, 2004, S. 72–83

[Domschke und Scholl 2005] Domschke, W.; Scholl, A.: *Grundlagen der Betriebswirtschaftslehre: Eine Einführung aus entscheidungsorientierter Sicht*. 3. Auflage. Berlin/Heidelberg/New York: Springer, 2005

[Dondo und Cerdá 2009] Dondo, R. G.; Cerdá, J.: A hybrid local improvement algorithm for large-scale multi-depot vehicle routing problems with time windows. In: *Computers & Chemical Engineering* 33 (2009), Nr. 2, S. 513–530

[Dongarra 2011] Dongarra, J. J.: *Performance of Various Computers Using Standard Linear Equations Software*. 2011. – URL `http://netlib.org/benchmark/performance.pdf`. – Zugriffsdatum: 23.12.2012

[Dorronsoro et al. 2007] Dorronsoro, B.; Arias, D.; Luna, F.; Nebro, A.; Alba, E.: A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. In: Zelinka, I. (Hrsg.); Oplatková, Z. (Hrsg.); Orsoni, A. (Hrsg.): *Proceedings 21st European Conference on Modelling and Simulation*, 2007, S. 759–765

[Du et al. 2012] Du, P.; Weber, R.; Luszczek, P.; Tomov, S.; Peterson, G.; Dongarra, J.: From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. In: *Parallel Computing* 38 (2012), Nr. 8, S. 391–407

[Duncan 1990] Duncan, R.: A survey of parallel computer architectures. In: *IEEE Computer* 23 (1990), Nr. 2, S. 5–16

[Eiben et al. 1994] Eiben, A.; Raué, P.; Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: Davidor, Y. (Hrsg.); Schwefel, H.-P. (Hrsg.); Männer, R. (Hrsg.): *Parallel Problem Solving from Nature*. Berlin/Heidelberg: Springer, 1994, S. 78–87

[Ergun et al. 2006] Ergun, Ö.; Orlin, J.; Steele-Feldman, A.: Creating very large scale neighborhoods out of smaller ones by compounding moves. In: *Journal of Heuristics* 12 (2006), Nr. 1, S. 115–140

[Fischer 2010] Fischer, M.: *Nvidia präsentiert GeForce GTX 470 und GTX 480.* 2010. – URL `http://heise.de/-965390`. – Zugriffsdatum: 29.10.2012

[Fischer 2012] Fischer, M.: *Nvidia veröffentlicht finale Version von CUDA 5.0.* 2012. – URL `http://heise.de/-1728973`. – Zugriffsdatum: 23.12.2012

[Fischer und Stiller 2013] Fischer, M.; Stiller, A.: Ein Fächer Schnelles: Nvidias Tesla K20 gegen Intel Xeon Phi und AMD FirePro W9000. In: *c't* (2013), Nr. 2, S. 76–81

[Fisher und Jaikumar 1981] Fisher, M. L.; Jaikumar, R.: A generalized assignment heuristic for vehicle routing. In: *Networks* 11 (1981), Nr. 2, S. 109–124

[Flynn 1972] Flynn, M. J.: Some computer organizations and their effectiveness. In: *IEEE Transactions on Computers* 100 (1972), Nr. 9, S. 948–960

[Garcia et al. 1994] Garcia, B.-L.; Potvin, J.-Y.; Rousseau, J.-M.: A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. In: *Computers & Operations Research* 21 (1994), Nr. 9, S. 1025–1033

[Garland und Kirk 2010] Garland, M.; Kirk, D. B.: Understanding throughput-oriented architectures. In: *Communications of the ACM* 53 (2010), Nr. 11, S. 58–66

[Gehring und Homberger 1999] Gehring, H.; Homberger, J.: A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Miettinen, K. (Hrsg.); Mäkelä, M. (Hrsg.); Toivanen, J. (Hrsg.): *Proceedings of EUROGEN99.* 1999, S. 57–64

[Gendreau und Potvin 2005] Gendreau, M.; Potvin, J.-Y.: Metaheuristics in combinatorial optimization. In: *Annals of Operations Research* 140 (2005), Nr. 1, S. 189–213

[Gendreau und Potvin 2010] Gendreau, M.; Potvin, J.-Y.: Tabu search. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics.* New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 41–59

[Gillett und Miller 1974] Gillett, B. E.; Miller, L. R.: A heuristic algorithm for the vehicle-dispatch problem. In: *Operational Research* 22 (1974), Nr. 2, S. 340–349

[Glover 1977] Glover, F.: Heuristics for integer programming using surrogate constraints. In: *Decision Sciences* 8 (1977), Nr. 1, S. 156–166

[Glover 1986] Glover, F.: Future paths for integer programming and links to artificial intelligence. In: *Computers & Operations Research* 13 (1986), Nr. 5, S. 533–549

[Glover 1989] Glover, F.: Tabu search - Part I. In: *OSRA Journal on Computing* 1 (1989), Nr. 3, S. 190–206

[Glover 1990] Glover, F.: Tabu search - Part II. In: *OSRA Journal on Computing* 2 (1990), Nr. 1, S. 4–32

[Golden et al. 1997] Golden, B. L.; Wasil, E. A.; Kelly, J. P.; Chao, I.-M.: *The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results.* 1997

[Golden et al. 1998] Golden, B. L.; Wasil, E. A.; Kelly, J. P.; Chao, I.-M.: The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic, T. G. (Hrsg.); Laporte, G. (Hrsg.): *Fleet Management and Logistics.* Boston: Kluwer, 1998, S. 33–56

[GPUReview o.J.] GPUReview: *nVidia GeForce 9500 GT PCI-E Video Card - Reviews, Specifications, and Pictures - GPUReview.com.* o.J.. – URL `http://www.gpureview.com/GeForce-9500-GT-PCI-E-card-574.html`. – Zugriffsdatum: 28.12.2012

[Gregg und Hazelwood 2011] Gregg, C.; Hazelwood, K.: Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In: *IEEE International Symposium on Performance Analysis of Systems and Software*, 2011, S. 134–144

[Groër et al. 2010] Groër, C.; Golden, B.; Wasil, E.: A library of local search heuristics for the vehicle routing problem. In: *Mathematical Programming Computation* 2 (2010), Nr. 2, S. 79–101

[Groër et al. 2011] Groër, C.; Golden, B.; Wasil, E.: A parallel algorithm for the vehicle routing problem. In: *INFORMS Journal on Computing* 23 (2011), Nr. 2, S. 315–330

[Gustafson 1988] Gustafson, J. L.: Reevaluating Amdahl's law. In: *Communications of the ACM* 31 (1988), Nr. 5, S. 532–533

[Hansen und Mladenović 2001] Hansen, P.; Mladenović, N.: Variable neighborhood search: Principles and applications. In: *European Journal of Operational Research* 130 (2001), Nr. 3, S. 449–467

[Hansen und Mladenović 2003] Hansen, P.; Mladenović, N.: Variable neighborhood search. In: Glover, F. (Hrsg.); Kochenberger, G. A. (Hrsg.): *Handbook of Metaheuristics.* Boston: Kluwer Academic Publishers, 2003, S. 145–184

[Hansen et al. 2010] Hansen, P.; Mladenović, N.; Brimberg, J.; Pérez, J. A. M.: Variable neighborhood search. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics.* New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 61–86

[Harris o.J.] Harris, M.; Nvidia (Hrsg.): *Optimizing Parallel Reduction in CUDA.* o.J.. – URL `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf`. – Zugriffsdatum: 28.12.2012

[Hertz und Kobler 2000] Hertz, A.; Kobler, D.: A framework for the description of evolutionary algorithms. In: *European Journal of Operational Research* 126 (2000), Nr. 1, S. 1–12

[Hillis und Steele 1986] Hillis, W. D.; Steele, G. L.: Data parallel algorithms. In: *Communications of the ACM* 29 (1986), Nr. 12, S. 1170–1183

[Ho und Gendreau 2006] Ho, S.; Gendreau, M.: Path relinking for the vehicle routing problem. In: *Journal of Heuristics* 12 (2006), S. 55–72

[Holland 1975] Holland, J. H.: *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press, 1975

[Intel 2011] Intel: *Processors: Intel microprocessor export compliance metrics.* 2011. – URL `http://www.intel.com/support/processors/sb/cs-023143.htm`. – Zugriffsdatum: 28.10.2011

[Intel o.J.] Intel: *Intel Core^{TM}2 Duo Processor E7200 (3M Cache, 2.53 GHz, 1066 MHz FSB).* o.J.. – URL `http://ark.intel.com/products/35348/Intel-Core2-Duo-Processor-E7200-3M-Cache-2_53-GHz-1066-MHz-FSB`. – Zugriffsdatum: 29.10.2012

[Irnich 2008a] Irnich, S.: A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. In: *INFORMS Journal on Computing* 20 (2008), Nr. 2, S. 270–287

[Irnich 2008b] Irnich, S.: Resource extension functions: properties, inversion, and generalization to segments. In: *OR Spectrum* 30 (2008), Nr. 1, S. 113–148

[Irnich et al. 2006] Irnich, S.; Funke, B.; Grünert, T.: Sequential search and its application to vehicle-routing problems. In: *Computers & Operations Research* 33 (2006), Nr. 8, S. 2405–2429

[Janiak et al. 2008] Janiak, A.; Janiak, W.; Lichtenstein, M.: Tabu search on GPU. In: *Journal of Universal Computer Science* 14 (2008), Nr. 14, S. 2416–2427

[Jin et al. 2011] Jin, J.; Crainic, T. G.; Løkketangen, A.: A guided cooperative parallel tabu search for the capacitated vehicle routing problem. In: *NIK.* 2011

[Jin et al. 2012] Jin, J.; Crainic, T. G.; Løkketangen, A.: A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. In: *European Journal of Operational Research* 222 (2012), Nr. 3, S. 441–451

[Johnson et al. 1989] Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; Schevon, C.: Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning. In: *Operations Research* 37 (1989), Nr. 3, S. 865–892

[Jozefowiez et al. 2002] Jozefowiez, N.; Semet, F.; Talbi, E.-G.: Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem. In: Guervós, J. (Hrsg.); Adamidis, P. (Hrsg.); Beyer, H.-G. (Hrsg.); Schwefel, H.-P. (Hrsg.); Fernández-Villacañas, J.-L. (Hrsg.): *Parallel Problem Solving from Nature.* Berlin/Heidelberg: Springer, 2002, S. 271–280

[Karimi et al. 2010] Karimi, K.; Dickson; Neil G.; Hamze, F.: *A performance comparison of CUDA and OpenCL.* 2010. – URL `http://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf`. – Zugriffsdatum: 29.11.2012

[Karp und Flatt 1990] Karp, A. H.; Flatt, H. P.: Measuring parallel processor performance. In: *Communications of the ACM* 33 (1990), Nr. 5, S. 539–543

[Kim und Bond 2009] Kim, H.; Bond, R.: Multicore software technologies. In: *IEEE Signal Processing Magazine* 26 (2009), Nr. 6, S. 80–89

[Kirk und Hwu 2010] Kirk, D.; Hwu, W.: *Programming Massively Parallel Processors: A Hands-On Approach (Applications of GPU Computing Series).* Burlington: Elsevier Inc., 2010

[Kirkpatrick et al. 1983] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.: Optimization by simulated annealing. In: *Science* 220 (1983), Nr. 4598, S. 671–680

[Kwon et al. 2007] Kwon, Y.-J.; Kim, J.-G.; Seo, J.; Lee, D.-H.; Kim, D.-S.: A tabu search algorithm using the voronoi diagram for the capacitated vehicle routing problem. In: *International Conference on Computational Science and its Applications*, 2007, S. 480–488

[Kytöjoki et al. 2007] Kytöjoki, J.; Nuortio, T.; Bräysy, O.; Gendreau, M.: An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. In: *Computers & Operations Research* 34 (2007), Nr. 9, S. 2743–2757

[Laporte 2009] Laporte, G.: Fifty years of vehicle routing. In: *Transportation Science* 43 (2009), Nr. 4, S. 408–416

[Laporte und Semet 2002] Laporte, G.; Semet, F.: Classical heuristics for the capacitated VRP. In: Toth, P. (Hrsg.); Vigo, D. (Hrsg.): *The Vehicle Routing Problem.* Philadelphia: Society for Industrial and Applied Mathematics, 2002, S. 109–128

[Lee et al. 2010] Lee, C.-Y.; Lee, Z.-J.; Lin, S.-W.; Ying, K.-C.: An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. In: *Applied Intelligence* 32 (2010), Nr. 1, S. 88–95

[Li et al. 2005] Li, F.; Golden, B.; Wasil, E.: Very large-scale vehicle routing: new test problems, algorithms, and results. In: *Computers & Operations Research* 32 (2005), Nr. 5, S. 1165–1179

[Li et al. 2011] Li, J.; Lv, X.; Liu, L.: A parallel genetic algorithm with GPU accelerated for large-scale MDVRP in emergency logistics. In: *International Conference on Computational Science and Engineering*, 2011, S. 602–605

[Lin et al. 2009] Lin, S.-W.; Lee, Z.-J.; Ying, K.-C.; Lee, C.-Y.: Applying hybrid meta-heuristics for capacitated vehicle routing problem. In: *Expert Systems with Applications* 36 (2009), Nr. 2, S. 1505–1512

[Luong et al. 2011] Luong, T.; Melab, N.; Talbi, E.-G.: GPU-based multi-start local search algorithms. In: Coello, C. (Hrsg.): *Learning and Intelligent Optimization.* Berlin/Heidelberg: Springer, 2011, S. 321–335

[Luong et al. 2013] Luong, T.; Melab, N.; Talbi, E.-G.: GPU computing for parallel local search metaheuristic algorithms. In: *IEEE Transactions on Computers* 62 (2013), Nr. 1, S. 173–185

[Luong et al. 2012] Luong, T.; Taillard, E.; Melab, N.; Talbi, E.-G.: Parallelization strategies for hybrid metaheuristics using a single GPU and multi-core resources. In: Coello, C. (Hrsg.);

Cutello, V. (Hrsg.); Deb, K. (Hrsg.); Forrest, S. (Hrsg.); Nicosia, G. (Hrsg.); Pavone, M. (Hrsg.): *Parallel Problem Solving from Nature*. Berlin/Heidelberg: Springer, 2012, S. 368–377

[Marinakis 2012] Marinakis, Y.: multiple phase neighborhood search-GRASP for the capacitated vehicle routing problem. In: *Expert Systems with Applications* 39 (2012), Nr. 8, S. 6807–6815

[Marinakis und Marinaki 2010] Marinakis, Y.; Marinaki, M.: A hybrid genetic – particle swarm optimization algorithm for the vehicle routing problem. In: *Expert Systems with Applications* 37 (2010), Nr. 2, S. 1446–1455

[Marinakis und Migdalas 2007] Marinakis, Y.; Migdalas, A.: Annotated bibliography in vehicle routing. In: *Operational Research* 7 (2007), Nr. 1, S. 27–46

[McClanahan 2010] McClanahan, C.: *History and Evolution of GPU Architecture*. 2010. – URL `http://mcclanahoochie.com/blog/wp-content/uploads/2011/03/gpu-hist-paper.pdf`. – Zugriffsdatum: 1.10.2012

[McCool 2008] McCool, M. D.: Scalable programming models for massively multicore processors. In: *Proceedings of the IEEE* 96 (2008), Nr. 5, S. 816–831

[Mester und Bräysy 2007] Mester, D.; Bräysy, O.: Active-guided evolution strategies for large-scale capacitated vehicle routing problems. In: *Computers & Operations Research* 34 (2007), Nr. 10, S. 2964–2975

[Mester et al. 2007] Mester, D.; Bräysy, O.; Dullaert, W.: A multi-parametric evolution strategies algorithm for vehicle routing problems. In: *Expert Systems with Applications* 32 (2007), Nr. 2, S. 508–517

[Metropolis et al. 1953] Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E.: Equation of state calculations by fast computing machines. In: *The Journal of Chemical Physics* 21 (1953), Nr. 6, S. 1087–1092

[Mladenović und Hansen 1997] Mladenović, N.; Hansen, P.: Variable neighborhood search. In: *Computers & Operations Research* 24 (1997), Nr. 11, S. 1097–1100

[Moscato und Cotta 2010] Moscato, P.; Cotta, C.: A modern introduction to memetic algorithms. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics*. New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 141–183

[Müller-Merbach 1992] Müller-Merbach, H.: *Operations Research: Methoden und Modelle der Optimalplanung*. 10. Nachdr. der 3., durchges. Auflage. München: Vahlen, 1992

[Nagata 2007] Nagata, Y.: Edge assembly crossover for the capacitated vehicle routing problem. In: Cotta, C. (Hrsg.); Hemert, J. van (Hrsg.): *Evolutionary Computation in Combinatorial Optimization*. Berlin/Heidelberg: Springer, 2007, S. 142–153

[Nagata und Bräysy 2008] Nagata, Y.; Bräysy, O.: Efficient local search limitation strategies for vehicle routing problems. In: Hemert, J. van (Hrsg.); Cotta, C. (Hrsg.): *Evolutionary Computation in Combinatorial Optimization*. Berlin/Heidelberg: Springer, 2008, S. 48–60

[Nagata und Bräysy 2009] Nagata, Y.; Bräysy, O.: Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. In: *Networks* 54 (2009), Nr. 4, S. 205–215

[Nashed et al. 2012] Nashed, Y.; Mesejo, P.; Ugolotti, R.; Dubois-Lacoste, J.; Cagnoni, S.: A comparative study of three gpu-based metaheuristics. In: Coello, C. (Hrsg.); Cutello, V. (Hrsg.); Deb, K. (Hrsg.); Forrest, S. (Hrsg.); Nicosia, G. (Hrsg.); Pavone, M. (Hrsg.): *Parallel Problem Solving from Nature* Bd. LNCS 7492. Berlin/Heidelberg: Springer, 2012, S. 398–407

[Nickolls und Dally 2010] Nickolls, J.; Dally, W. J.: The GPU computing era. In: *IEEE Micro* 30 (2010), Nr. 2, S. 56–69

[Nickolls und Kirk 2011] Nickolls, J.; Kirk, D.: Appendix A: Graphics and computing GPUs. In: Patterson, D. A. (Hrsg.); Hennessy, J. L. (Hrsg.): *Rechnerorganisation und Rechnerentwurf.* München: Oldenbourg, 2011, S. A.1–A.77

[Nikolaev und Jacobson 2010] Nikolaev, A. G.; Jacobson, S. H.: Simulated annealing. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics.* New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 1–39

[Nourani und Andresen 1998] Nourani, Y.; Andresen, B.: A comparison of simulated annealing cooling strategies. In: *Journal of Physics A: Mathematical and General* 31 (1998), Nr. 41, S. 8373–8385

[Nvidia 2009] Nvidia: *NVIDIA's Next Generation CUDA Compute Architecture: Fermi.* 2009. – URL `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`. – Zugriffsdatum: 3.10.2012

[Nvidia 2011a] Nvidia: *CUDA C Best Practices Guide (v4.0).* 2011

[Nvidia 2011b] Nvidia: *NVIDIA CUDA C Programming Guide (Version 4.0).* 2011

[Nvidia 2011c] Nvidia: *Tesla M-Class GPU Computing Modules Accelerating Science.* 2011. – URL `http://www.nvidia.com/docs/IO/105880/DS-Tesla-M-Class-Aug11.pdf`. – Zugriffsdatum: 14.12.2012

[Nvidia 2012] Nvidia: *NVIDIA CUDA C Programming Guide (Version 4.2).* 2012

[Or 1976] Or, I.: *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking.* Evanston, Northwestern University, Dissertation, 1976

[Osman und Laporte 1996] Osman, I.; Laporte, G.: Metaheuristics: A bibliography. In: *Annals of Operations Research* 63 (1996), Nr. 5, S. 511–623

[Pander 2012] Pander, J.: *Spritspartechnik für LKW: Wir sparn, sparn, sparn auf der Autobahn.* 2012. – URL `http://www.spiegel.de/auto/aktuell/technische-innovationen-sollen-lkw-sparsamer-und-sauberer-machen-a-856491.html`. – Zugriffsdatum: 18.12.2012

[Pereira et al. 2002] Pereira, F.; Tavares, J.; Machado, P.; Costa, E.: GVR: a new genetic representation for the vehicle routing problem. In: O'Neill, M. (Hrsg.); Sutcliffe, R. (Hrsg.); Ryan, C. (Hrsg.); Eaton, M. (Hrsg.); Griffith, N. (Hrsg.): *Artificial Intelligence and Cognitive Science*. Berlin/Heidelberg: Springer, 2002, S. 285–320

[Pisinger und Ropke 2007] Pisinger, D.; Ropke, S.: A general heuristic for vehicle routing problems. In: *Computers & Operations Research* 34 (2007), Nr. 8, S. 2403–2435

[Potvin und Rousseau 1995] Potvin, J.-Y.; Rousseau, J.-M.: An exchange heuristic for routeing problems with time windows. In: *The Journal of the Operational Research Society* 46 (1995), Nr. 12, S. 1433–1446

[Prins 2004] Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. In: *Computers & Operations Research* 31 (2004), Nr. 12, S. 1985–2002

[Prins 2009] Prins, C.: A GRASP × evolutionary local search hybrid for the vehicle routing problem. In: Pereira, F. (Hrsg.); Tavares, J. (Hrsg.): *Bio-inspired Algorithms for the Vehicle Routing Problem*. Berlin/Heidelberg: Springer, 2009, S. 35–53

[Proudfoot et al. 2001] Proudfoot, K.; Mark, W. R.; Tzvetkov, S.; Hanrahan, P.: A real-time procedural shading system for programmable graphics hardware. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, S. 159–170

[Ralphs et al. 2003] Ralphs, T.; Kopman, L.; Pulleyblank, W.; Trotter, L.: On the capacitated vehicle routing problem. In: *Mathematical Programming* 94 (2003), Nr. 2, S. 343–359

[Rauber und Rünger 2008] Rauber, T.; Rünger, G.: *Multicore: Parallele Programmierung*. Berlin/Heidelberg: Springer, 2008

[Rauber und Rünger 2010] Rauber, T.; Rünger, G.: *Parallel Programming for Multicore and Cluster Systems*. Berlin: Springer, 2010

[Rauber und Rünger 2012] Rauber, T.; Rünger, G.: *Parallele Programmierung*. 3. Auflage. Berlin/Heidelberg: Springer, 2012

[Rechenberg 1973] Rechenberg, I.: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Fommann-Holzboog, 1973

[Reeves 2010] Reeves, C. R.: Genetic algorithms. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics*. New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 109–139

[Reimann et al. 2003] Reimann, M.; Doerner, K.; Hartl, R. F.: Analyzing a unified ant system for the VRP and some of its variants. In: *Proceedings of the international conference on Applications of evolutionary computing*, Springer, 2003, S. 300–310

[Reimann et al. 2004] Reimann, M.; Doerner, K.; Hartl, R. F.: D-ants: Savings based ants divide and conquer the vehicle routing problem. In: *Computers & Operations Research* 31 (2004), Nr. 4, S. 563–591

[Rothlauf 2011] Rothlauf, F.: *Design of modern heuristics: Principles and application.* Berlin/New York: Springer, 2011

[Sanders und Kandrot 2010] Sanders, J.; Kandrot, E.: *CUDA by Example: An Introduction to General-Purpose GPU programming.* Upper Saddle River: Addison-Wesley, 2010

[Schneider und Puchta 2010] Schneider, J. J.; Puchta, M.: Investigation of acceptance simulated annealing - A simplified approach to adaptive cooling schedules. In: *Physica A: Statistical Mechanics and its Applications* 389 (2010), Nr. 24, S. 5822–5831

[Schneider 2012] Schneider, M.: *New Challenges in Time-Definite Vehicle Routing.* Kaiserslautern, Technische Universität Kaiserslautern, Dissertation, 2012

[Schulz 2013] Schulz, C.: Efficient local search on the GPU—Investigations on the vehicle routing problem. In: *Journal of Parallel and Distributed Computing* 73 (2013), Nr. 1, S. 14–31

[Silberholz und Golden 2010] Silberholz, J.; Golden, B.: Comparison of metaheuristics. In: Gendreau, M. (Hrsg.); Potvin, J.-Y. (Hrsg.): *Handbook of Metaheuristics.* New York/Dordrecht/Heidelberg/London: Springer, 2010, S. 625–640

[Sodan et al. 2010] Sodan, A. C.; Machina, J.; Deshmeh, A.; Macnaughton, K.; Esbaugh, B.: Parallelism via multithreaded and multicore CPUs. In: *Computer* 43 (2010), Nr. 3, S. 24–32

[Solomon 1987] Solomon, M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. In: *Operations Research* 35 (1987), Nr. 2, S. 254–265

[Stiller 2010] Stiller, A.: *Top500 der Supercomputer: Asien dominiert die Spitze dank Nvidia.* 2010. – URL `http://heise.de/-1136052`. – Zugriffsdatum: 28.10.2011

[Stiller 2012] Stiller, A.: *SC12: Intel bringt Coprozessor Xeon Phi offiziell heraus.* 2012. – URL `http://heise.de/-1747942`. – Zugriffsdatum: 23.12.2012

[Suman und Kumar 2005] Suman, B.; Kumar, P.: A survey of simulated annealing as a tool for single and multiobjective optimization. In: *Journal of the Operational Research Society* 57 (2005), Nr. 10, S. 1143–1160

[Szeto et al. 2011] Szeto, W. Y.; Wu, Y.; Ho, S. C.: An artificial bee colony algorithm for the capacitated vehicle routing problem. In: *European Journal of Operational Research* 215 (2011), Nr. 1, S. 126–135

[Taillard 1993] Taillard, É.: Parallel iterative search methods for vehicle routing problems. In: *Networks* 23 (1993), Nr. 8, S. 661–673

[Taillard et al. 1997] Taillard, É.; Badeau, P.; Gendreau, M.; Guertin, F.; Potvin, J.-Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. In: *Transportation Science* 31 (1997), Nr. 2, S. 170–186

[Tarantilis 2005] Tarantilis, C. D.: Solving the vehicle routing problem with adaptive memory programming methodology. In: *Computers & Operations Research* 32 (2005), Nr. 9, S. 2309–2327

[Thompson et al. 2002] Thompson, C. J.; Hahn, S.; Oskin, M.: Using modern graphics architectures for general-purpose computing: a framework and analysis. In: *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, 2002, S. 306–317

[Toth und Vigo 2002a] Toth, P.; Vigo, D.: Models, relaxations and exact approaches for the capacitated vehicle routing problem. In: *Discrete Applied Mathematics* 123 (2002), Nr. 1-3, S. 487–512

[Toth und Vigo 2002b] Toth, P. (Hrsg.); Vigo, D. (Hrsg.): *The Vehicle Routing Problem*. Philadelphia: Society for Industrial and Applied Mathematics, 2002

[Toth und Vigo 2003] Toth, P.; Vigo, D.: The granular tabu search and its application to the vehicle-routing problem. In: *INFORMS Journal on Computing* 15 (2003), Nr. 4, S. 333–346

[Tsutsui und Ghosh 1998] Tsutsui, S.; Ghosh, A.: A study on the effect of multi-parent recombination in real coded genetic algorithms. In: *Evolutionary Computation Proceedings*, 1998, S. 828–833

[Turky 2011] Turky, A. M.: Adaptive temperature control of simulated annealing for solving capacitated vehicle routing problem. In: *International Conference on Telecommunication Technology and Applications*. 2011, S. 80–82

[Varanelli und Cohoon 1999] Varanelli, J. M.; Cohoon, J. P.: A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. In: *Computers & Operations Research* 26 (1999), Nr. 5, S. 481–503

[Varbanescu 2010] Varbanescu, A. L.: *On the effective parallel programming of multi-core processors*. Delft, Technische Universität Delft, Dissertation, 2010

[Vidal et al. 2011] Vidal, T.; Crainic, T. G.; Gendreau, M.; Lahrichi, N.; Rei, W.: A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. 2011 (CIRRELT-2011-05). Working Paper

[Wendt 1995] Wendt, O.: *Tourenplanung durch Einsatz naturanaloger Verfahren: Integration von genetischen Algorithmen und Simulated Annealing*. Wiesbaden: Gabler, 1995

[Weyland et al. 2013] Weyland, D.; Montemanni, R.; Gambardella, L. M.: A metaheuristic framework for stochastic combinatorial optimization problems based on GPGPU with a case study on the probabilistic traveling salesman problem with deadlines. In: *Journal of Parallel and Distributed Computing* 73 (2013), Nr. 1, S. 74–85

[Wittenbrink et al. 2011] Wittenbrink, C. M.; Kilgariff, E.; Prabhu, A.: Fermi GF100 GPU architecture. In: *IEEE Micro* 31 (2011), Nr. 2, S. 50–59