Using TEAMWORK for the Distribution of Approximately Solving the Traveling Salesman Problem with Genetic Algorithms

Jörg Denzinger, Stephan Scholz
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany
E-mail: {denzinge|stscholz}@informatik.uni-kl.de

April 25, 1997

Abstract

We present a distributed system, DOTT, for approximately solving the Traveling Salesman Problem (TSP) based on the TEAMWORK method. So-called experts and specialists work independently and in parallel for given time periods. For TSP, specialists are tour construction algorithms and experts use modified genetic algorithms in which after each application of a genetic operator the resulting tour is locally optimized before it is added to the population. After a given time period the work of each expert and specialist is judged by a referee. A new start population, including selected individuals from each expert and specialist, is generated by the supervisor, based on the judgments of the referees. Our system is able to find better tours than each of the experts or specialists working alone. Also results comparable to those of single runs can be found much faster by a team.

1 Introduction

The Traveling Salesman Problem (TSP) is a well-known optimization problem. Given a set of n cities, the TSP is finding the shortest round-trip, such that each city is visited exactly once. Obviously the TSP can be interpreted easily in a geographical way, but moreover in a theoretical way; a lot of problems can actually be transformed into Traveling Salesman Problems.

Although the TSP is easily explained in its idea, it belongs to the set of NP-hard tasks. It is often referred to as a standard example of NP-hard problems. Therefore one is often satisfied with an approximate solution, i.e. a solution that is only a few parts per thousand worse than the best solution, as long as this approximate solution can be computed in an acceptable time.

Promising results for solving optimization problems in an approximate way have been achieved with genetic algorithms. Similar to Darwin's principle of natural evolution, a set of *individuals*, called *population*, is treated by two genetic operators: *mutation* and *crossover*. While mutation stands for manipulating the *genes* of an individual, crossover means combining the genes of two parents in order to create a new individual. A *fitness function* determines the quality of an individual; the ones with the least fitness are being removed from the population, in order to advance the whole evolution process ("survival of the fittest").

For the TSP, an individual is simply a tour, which can either be manipulated by a mutation step or combined with another individual in order to create a new tour (crossover). Though genetic algorithms have shown good results in experiments, they still need quite some time until good individuals come up. In order to overcome this problem, we tried an approach of applying a local optimization heuristic after each genetic operation. This technique is quite similar to the normalization of new formulas that is part of many theorem proving programs. In theorem proving for each new generated formula subsumption tests are performed and other formulas are used to simplify it before the formula is put into the data base of known formulas. In genetic algorithms each new individual is locally optimized before it becomes part of the population. Experiments, not only made by us, have shown that indeed this helps to improve the power of genetic algorithms very much.

In this report we will present a distributed system for solving the TSP. Basis is our TEAMWORK method for the distribution of search processes that are based on *search by extension and focus*. Processes employing search by extension and focus use sets of facts as representation of the search state and typically allow to each state a large number of transitions to other states that extend the set of facts of the state by new facts. In order to control such search processes, a good extension has to be selected, which is the task of a so-called *focus function*. We will show that search using genetic algorithms can be seen as such a search by extension and focus.

The main problems of such search processes are that there is a wide variety of focus functions but nevertheless there are problems that are not solved well by any of the focus functions implemented in a search system. So a dynamical adaption of the search

focus would be useful. But even if there is a well suited focus function implemented, the problem of finding it remains.

Our TEAMWORK method solves these problems by a distribution concept that realizes cooperation and competition of several focus functions in form of so-called experts. The experts and perhaps some specialists work in parallel (on different computing nodes) for a given period of time on solving the given problem. Experts employ search by extension and focus, but they all use different focus functions, thus exploring different parts of the search space. Specialists generate facts by other means or can compute other necessary data.

After such a working period all experts and specialists are judged by referees that compute a measure of success for the whole work accomplished by an expert/specialist and select very good facts that promise to contribute to a solution of the given problem. These results are communicated to the supervisor that generates a start state for a new working period out of the search state of the expert with the best measure of success and the selected good facts. The supervisor can also exchange experts and specialists with bad measures of success and this way provides the system with planning and self-adaption capabilities.

TEAMWORK has already been applied to several problems that can be solved by search by extension and focus. Our systems for equational theorem proving, theorem proving using condensed detachment, or solving time tabling problems showed that TEAMWORK allows for synergetic effects that result in a much faster generation of good solutions and also in better solutions compared to sequential runs. In this paper we will demonstrate that this is also the case for solving the TSP using genetic algorithms. Again, our system DOTT finds solutions comparable to the best solutions of the single genetic algorithms much faster and is also able to generate better solutions than all the single experts are capable of.

This article is organized as follows: in section 2 we introduce the basic concepts of genetic algorithms and the TEAMWORK method. In section 3 we define the TSP and present a way of using the TEAMWORK method for solving it: the software system DOTT. Section 4 gives the results of experiments that have been made in order to test DOTT, to find out whether the promising ideas show good results in reality. Finally, we give conclusions and possibilities of future work in section 5.

2 Genetic Algorithms and TEAMWORK

In this section, we give an introduction to genetic algorithms in general, especially in relation to our definition of the search process called "search by extension and focus". Furthermore we present an approach to distribute this search process, namely the TEAMWORK method.

2.1 Genetic Algorithms

John H. Holland was the first one to try a new approach for solving optimization problems (see [Ho92]). The idea of natural evolution was the inspiration for these so-called *evolution strategies* and *genetic algorithms*. A set of *individuals*, called *population*, is manipulated by two main operators: *mutation* and *crossover*. A *fitness function* decides, which of the individuals have a good resp. poor quality, and the best individuals according to this fitness function are selected for the next generation. Obviously this works in the same way as Darwin's principle of the *survival of the fittest*. Great emphasis is put on the non-determinism; a random number generator is used in the selection of the individuals the genetic operators are applied to (but their fitness plays also a role, again). Therefore a run of a genetic algorithm is generally not reproducible. We will explain the two genetic operators in more detail. Individuals of the population have certain attributes which are called *genes*. In a mutation step, some of the genes of an individual are manipulated; in a crossover step, the genes of the parents are mixed in the way that a new individual is created.

Crossover For a crossover step, two parent individuals have to be selected first. With genetic algorithms, this happens in a probabilistic way; the individuals are randomly chosen for performing a crossover step. However, a probabilistic distribution is used in order to prefer the fitter individuals.

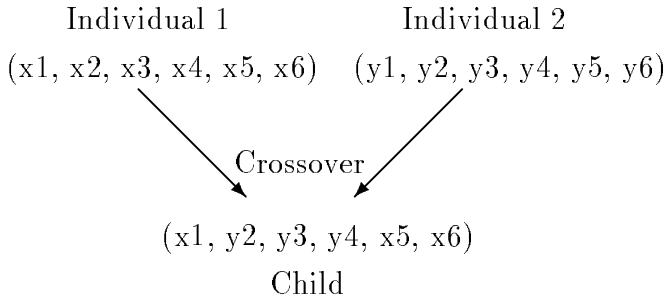

Figure 1: An example of a crossover step

As can be seen in Figure 1, individuals in this example have six genes; the child is created by combining the genes of the parents. As will be shown later, crossover is not always as easy as in this example. In more complicated cases, one must also make sure that the child is indeed a correct individual according to the specification.

Mutation For a mutation step, one individual is selected from the population and its genes are manipulated in a way that a new individual is created. Again it is important that the choosing of the individual is done in a probabilistic way with some preference to fitter individuals.

Fitness function The fitness function is one of the most important factors for genetic algorithms. Because it determines the quality of each individual, it actually

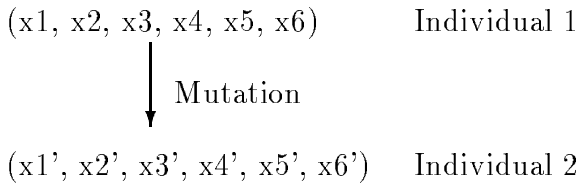

Figure 2: An example of a mutation step

controls the whole optimization process. Optimization methods have a tendency of entering local optima if the system moves close to one. This way it often happens that a better solution than the locally best one cannot be found anymore. Therefore it can make sense to select individuals that do not seem to be helpful for the optimization process at once, but lead to new possibilities and ideas. In that way, the system may escape a local optimum and reach the global optimum afterwards. It is very hard to say which individuals actually carry the potential of finding the best solution, i.e. the global optimum. Obviously the fitness function depends on the given optimization problem and one has to put a lot of effort in finding a good one in order to achieve satisfying results.

Both mutation and crossover operate on existing individuals. Therefore an initial population has to be created. Mutation and crossover produce new individuals, i.e. the population grows; the fitness function is responsible for the selection. It decides which individuals are being removed from the population.

Figure 3 gives an overview of how genetic algorithms work. It has been shown in experiments that genetic algorithms tend to be quite slow, because a lot of the produced individuals have a poor quality. In order to speed up the process, one may improve the newly created individuals by applying a (conventional) optimization step after each genetic operation. For efficiency reasons, this optimization step has to be a local one, i.e. a step that will generate an individual (to the newly created one) that is locally optimal (see Figure 3). As can be seen in experiments, algorithms run much faster this way.

In the classical approach of genetic algorithms, genes are simply binary numbers; manipulation of a gene therefore is accomplished easily by inverting the gene. This is very close to the biological view (genes have four different states there). Crossover, on the other hand, means somehow mixing the two gene sets of the parents in order to create a new individual, for example randomly or by choosing a position in the sequence of genes so that the new individual consists of the genes before the position of one parent and after the position of the other one. It has been shown, however, that it is better to have a more general view on genetic algorithms and to take advantage of the given problem. Crossover and mutation are different for the specific problems then. The main advantage of this approach is that a huge speed-up is achieved. One can use the existing knowledge of a given problem for the definition of crossover and mutation.

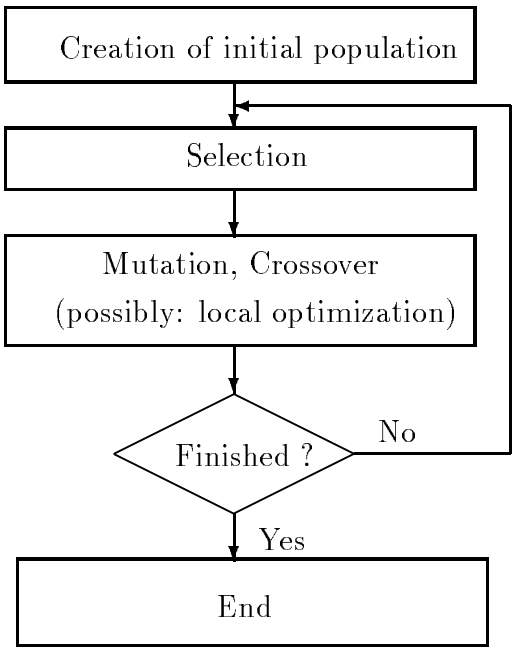

Figure 3: Genetic algorithms in general

2.2 Search by Extension and Focus

Obviously genetic algorithms are used for search processes. We will now show how genetic algorithms can be formally described by using search by extension and focus.

In literature search processes are described by *states* and *transition rules* between states. A useful classification of search processes is based on the representation of states. There are several groups of search processes, but for our purposes a distinction in only two groups suffices. Members of the first group need explicit information about the history of the process while members of the second one do not need to represent this information explicitly.

Search processes of the first group are often based on such principles as dividing a problem into subproblems and therefore use trees or directed graphs as representations of the search state. Search processes of the second group use sets of "results" as representations of the state. Note that there may be processes of both groups that can be used to solve a given search problem. We will give in the following a formal definition of the processes of the second group. We will also see that the search processes that are defined by genetic algorithms belong to the second group: search by extension and focus.

Definition 2.1 (Search by extension and focus)
Search by extension and focus *is described by a 4-tuple* $(\mathcal{B},\Omega,\mathcal{I},S_0)$. *The set of possible facts* $\mathcal{B}$ *defines the possible states* S *of the search by* $S \in 2^{\mathcal{B}}$. Ω *is a predicate defined on* $\mathcal{B}$ *and used to describe a legal state* S *of the search by* $\Omega(s)$ *is true for all* $s \in S$. $\mathcal{I}$ *is a*

set of extension rules $A \to B$, $A,B \in 2^{\mathcal{B}}$. S_0 is called the start state of the search and has to be a legal state. We write $S \vdash_{\mathcal{I}} S'$ for states S and S', if there is a rule $A \to B \in \mathcal{I}$, such that $A \subseteq S$ and $S' = (S \,/\, A) \cup B$. A sequence $(S_0,S_1,\ldots,S_n)$ with $S_{i-1} \vdash_{\mathcal{I}} S_i$ for $i=1,\ldots,n$, is called a search derivation.

In our special case of genetic algorithms, the possible facts $\mathcal{B}$ are all valid individuals according to the specification. A state S of the search process is the set of individuals that have been found up to this point, i.e. the population at this special time. Obviously the state S_0 is the initial population. The set of extension rules $\mathcal{I}$ contains the actual genetic operations, i.e. crossover and mutation. The selection of the fittest individuals is included in $\mathcal{I}$ as well. One way is to include it into the extension rules to crossover and mutation by doing first such a genetic operation and then a selection step, i.e. deleting some individuals. Another way is to include a special extension rule "selection" that has an empty set B (i.e. an empty right side).

Typically, searching means applying a search process to an instance of the search problem. In our formal definition the search process is represented by $\mathcal{B}$ and $\mathcal{I}$, while the actual instance determines S_0 and Ω and also provides the goal of the search.

Definition 2.2 (Goal of a search)
Let $(\mathcal{B},\Omega,\mathcal{I},S_0)$ be a search by extension and focus and $g \in \mathcal{B}$ with $\Omega(g) = true$ the goal *of the search. A state containing g is called a* goal state. *The goal is* reachable, *if there is a search derivation $(S_0,S_1,\ldots,S_n)$ such that S_n is a goal state.*

Figure 4 shows a search derivation from the start state to a goal state. For each state of the search process there are different opportunities of moving on to a new state, i.e. there are different extensions. The main problem of a search is to find a (short) search derivation to a state that includes the goal. Note that there may be different solutions to a search problem that have the same quality. It is necessary to define the desired attributes of the goal in order to decide whether it has been reached or not. Once an individual with the desired characteristics is created, the search process is stopped ("the goal is reached").

There may be many possible extensions to a given state; for example in Figure 4, from S_0 the state S_1 was chosen (instead of S_1', S_1" etc.). Consequently there has to be a function that determines which extension should be chosen next, i.e. a function providing a *focus*. Typically, such a focus function associates with each pair (state, extension rule) a weight that rates the extension step.

Definition 2.3 (Focus function)
Let $(\mathcal{B},\Omega,\mathcal{I},S_0)$ be a search by extension and focus and g its goal. An injective function $f:2^{\mathcal{B}} \times \mathcal{I} \to \mathbb{Z}$ is called a focus function *and the derivation $(S_0,S_1,\ldots,S_i,\ldots)$ is produced by f, if for the extension $A_i \to B_i$ that produced state S_i we have $f(S_{i-1},A_i \to B_i) \leq f(S_{i-1},A \to B)$ for all $A \to B \in \mathcal{I}$.*

If one wants to allow only legal states in a search sequence, then the focus function only has to rate pairs $(S_{i-1},A \to B)$ that produce a legal state. In practise, often the

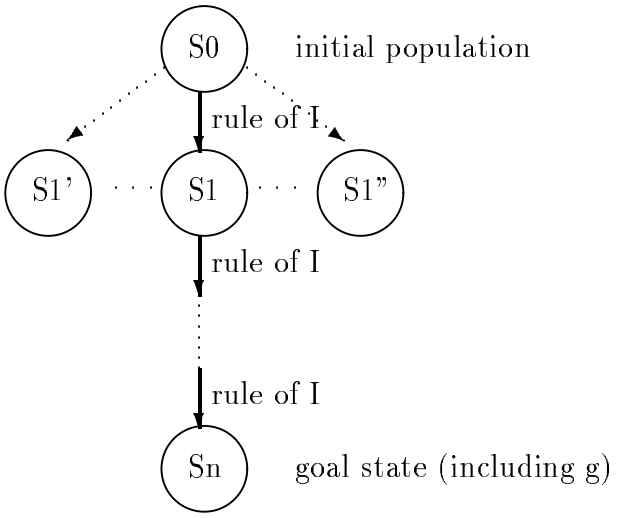

Figure 4: Search derivation

condition "injective" function is dropped and the decision between extension steps with equal f-value is made employing the FIFO-strategy. This way there are many focus functions that can be used, often too many to allow an automated decision which one to choose for a given problem instance. Another problem is that very often none of the implemented focus functions are good enough to solve a given problem instance in an acceptable time. These problems are solved by our TEAMWORK method.

For genetic algorithms, a focus function has to take into account two different aspects. First the actual genetic operator (mutation or crossover). Second, the possible deletion of individuals (selection). Both of these two tasks form an extension step, and the fitness function is used for each of them. However, the genetic operators work in a probabilistic way; the choosing of individuals for performing crossover resp. mutation as well as the operations themselves are done with the use of a random number generator (the selection, on the other hand, works in a deterministic way). Therefore the focus function reduces the set of possible extensions to a partial set (the set of all possible individuals that can be created by the genetic operations with possible selection afterwards). But which extension of this partial set will be actually used is decided randomly.

2.3 TEAMWORK for Distributed Search

The TEAMWORK method is our approach to distribute search by extension and focus. A system based on TEAMWORK has four types of components: experts, specialists, referees and a supervisor. The interaction between these components is organized as a cycle with three phases. In the **competition phase**, also called working phase, experts and specialists work independently on their tasks. In the **judgment phase**, the first part of a *team meeting*, referees judge the work of the experts and specialists

and select outstanding results and in the **cooperation phase**, the second part of a team meeting, the supervisor generates a new start state for the further search.

Experts work on solving the given problem instance by using search by extension and focus. Each expert uses a different focus function, thus generating different search sequences.

Definition 2.4 (Expert)
An expert *X is characterized by a focus function f_X: $2^{\mathcal{B}} \times \mathcal{I} \to \mathbb{Z}$. An expert starts cycle i with start state S_i. During cycle i an expert may be active, i.e. running on a processor, or not. By $\mathcal{E}xp$ we denote the set of all experts.*

For genetic algorithms, an expert takes a given population S_i and uses mutation resp. crossover along with selection to modify the individuals in order to achieve a new population (which can be used by the supervisor to construct the start population S_{i+1}). Therefore an expert only operates on a given set of individuals (the creation of an initial population has to be done by a specialist). The difference between experts can be achieved either by using different random number generators (this does not influence the selection process) or by using different fitness functions (which influences the whole search).

Specialists can also work on solving the problem instance, without being limited to using search by extension and focus, or they can generate data that helps controlling the search, or they can combine these two tasks.

Definition 2.5 (Specialist)
A specialist *Sp is a function f_{Sp}: $2^{\mathcal{B}} \to 2^{\mathcal{B}} \times$ message-set. It starts cycle i with the set S_i of facts and returns a set Res_{Sp} of facts with $\Omega(s) = true$ for all $s \in Res_{Sp}$ and it can also return a message out of a set of messages for the supervisor. During a cycle a specialist may be active or not. By $\mathcal{S}pec$ we denote the set of all specialists.*

Note that a specialist can produce its set Res_{Sp} by any correct means. The set of possible messages *message-set* has to be defined by the user. Each message of *message-set* has to be interpreted by the supervisor.

For genetic algorithms, it has to be considered that experts only operate on an existing population. Therefore it is necessary to have at least one specialist that creates an initial population (or part of it). In this case, the set S_i is empty while Res_{Sp} is the initial population. Moreover, specialists may be used for performing special tasks like calculating useful information (e.g. a bound in order to decide whether the search process should be stopped). In this case, the set Res_{Sp} is empty while the *message-set* contains this information.

Referees have two tasks: computing a measure of success for experts and some specialists and selecting outstanding results of experts and specialists. The results of both tasks are passed on to the supervisor. A referee of a specialist also reports the message of the specialist (if there is any).

Definition 2.6 (Referee)
A referee R *consists of two functions*

$meas_R$: $(2^{\mathcal{B}})^* \to \mathbb{Z}$ *and*

$selres_R$: $2^{\mathcal{B}} \to \mathcal{B}^{\leq k}$,

where $(2^B)^*$ *is a sequence of states and* k *the maximal number of results that may be selected by the referee. By* $\mathcal{R}ef$ *we denote the set of all referees.*

Generally, if a referee judges the success of an expert then it uses the whole search sequence produced by it as basis for its judgment. Since specialists can use totally different representations for their search states, the referee can only use the set Res_{Sp}. This is also the set $selres_R$ chooses from, while a referee of an expert uses its last search state as input for $selres_R$.

For genetic algorithms, however, the function $meas_R$ uses only the actual population of the expert for calculating a measure, since the whole search derivation is normally not stored. A simple $meas_R$ function uses the fitness function and calculates the mean fitness of the produced individuals in order to determine the quality of an expert's work. The $selres_R$ function also uses the actual population and selects at most k individuals for transmitting them to the supervisor. Note that this function may be used to avoid a local optimum e.g. by selecting individuals that are very different from each other in order to get a wide variety of individuals in the population.

The supervisor achieves the cooperation of experts and specialists by generating a new start state for the next cycle out of their results. But it also compares the success of the experts and specialists and selects the members of the team of the next cycle using the measures of success of the experts and specialists in prior cycles and further information provided by a long-term memory (see [DK94]). Also the messages of the specialists are used. As third task the supervisor determines the length of the next cycle using the same information.

Definition 2.7 (Supervisor)
The supervisor *computes after a cycle* i *the following three functions:*

$Comp$:$(2^{\mathcal{B}} \times \mathcal{B}^k \times \mathbb{Z})^n \to 2^{\mathcal{B}}$

Sel_i:$(\mathcal{E}xp \cup \mathcal{S}pec) \times \mathcal{R}ef \to \{0,1\}$

$Time_i$: $2^{\mathcal{B}} \times ((\mathcal{E}xp \cup \mathcal{S}pec) \times \mathcal{R}ef)^n \to I\!N$

where n *is the number of available processors.*

While the function $Comp$ that computes the new start state remains the same throughout a whole distributed search run –it simply uses the state produced by the expert with the best measure of success and adds the selected results of the other experts and the specialists to generate the start state– the functions Sel and $Time$ change from cycle to cycle, indicated by index i, because they have to take the results of the prior cycles and the messages of the specialists into account. Since there are only n processors available, we demand $\sum_{x \in \mathcal{E}xp \cup \mathcal{S}pec} Sel_i(x, \_) = n$, so that only n experts and specialists (with their referees) are active in each cycle.

In our case of genetic algorithms, the function *Comp* creates a new start population for the next cycle based on the selected individuals ($selres_R$) by the referees as well as on all the results of the winner (they are entirely used for the new start population).

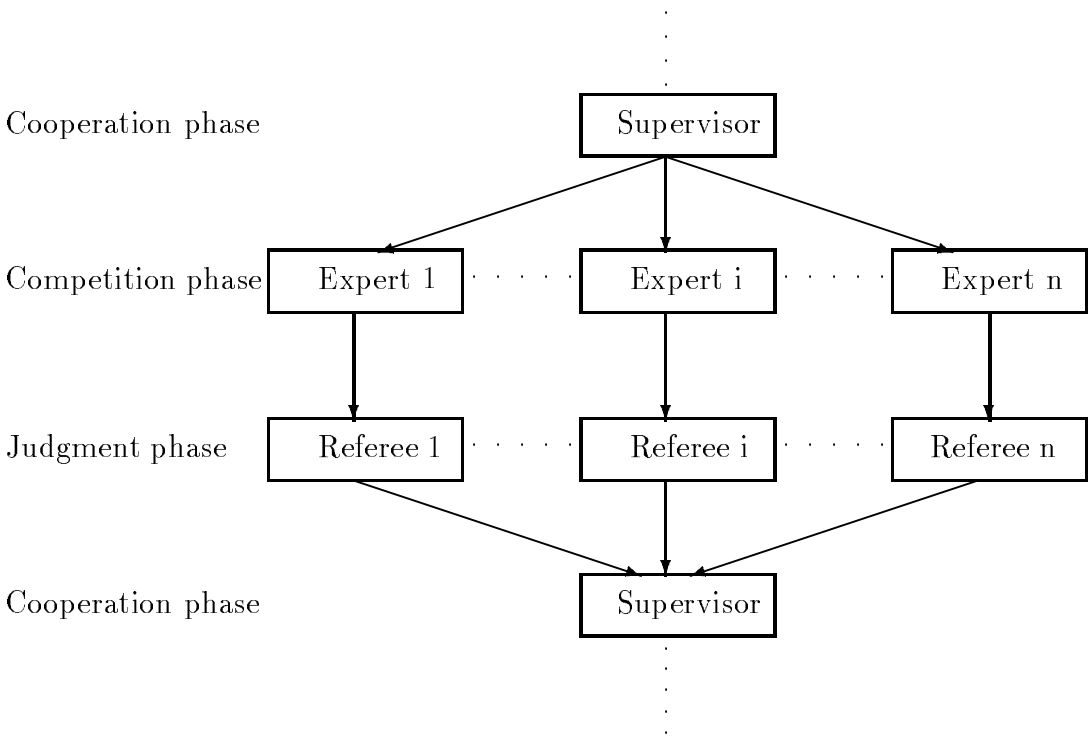

Figure 5: Overview of TEAMWORK

Figure 5 shows the whole distributed search process of TEAMWORK. Implementing the control cycle and the communication between the components requires more than knowledge about the search problem one wants to solve. It is possible to avoid some interprocessor communication by allowing components of different types to share a processor. Since an expert or specialist, a referee, or the supervisor never are active during the same time, one can implement them as one process having different modes (or an agent having different roles).

So, at the beginning of a distributed run, the process on one processor is in supervisor mode, receiving the problem instance. After generating a search state, the process sends this state to all other processors whose processes are in expert or specialist mode. Until the next team meeting the work of the supervisor is done and this process changes to expert or specialist mode.

When the end of a working phase is reached, each processor changes into referee mode. This way the referees can access all data of their experts/specialists without expensive communication. After the judgment phase the process that was in supervisor mode at the end of the last meeting changes back into supervisor mode and receives the measures of success of all experts and specialists (function $meas_r$). After the best

expert is determined, its process changes to supervisor mode. This new supervisor receives the full reports of the referees. The selected results of the other experts and the specialists can be integrated directly into the actual search state of the process in supervisor mode. After determining the members of the team in the next cycle the new start state is transmitted to the other processes and a new cycle begins.

By carefully choosing point-to-point connections between processors or broadcasting to all processors (when transmitting the new start state), one is able to achieve communication and control without much overhead. Our *TWlib* (see [DL96]) provides classes and methods that achieve this. Then the following synergetic effects occur, that allow a team to be much more effective than single experts that employ only one focus function.

- A focus function that is able to select many necessary extension steps on the way to a goal may nevertheless rate some few necessary extension steps very bad. If there is another focus function that rates the missing extensions good, then the TEAMWORK method allows the experts of these focus functions to cooperate, which results in infusing the missing results in the state of the first expert when they are found by the second one. So, this effect is based on the cooperation aspect of TEAMWORK.
- An expert may be able to generate a good search state, but then can not continue towards the goal. Another expert can, starting with this good search state, continue towards the goal. This change of focus functions produces search sequences that are much better than all sequences produced by one expert alone thus constituting a kind of "hyper"-heuristic. This effect is based on the competition aspect of TEAMWORK.

3 TSP and TEAMWORK

In this section we present our system DOTT (**D**istributed **O**ptimization of the **T**raveling Salesman Problem using **TEAMWORK**). Processes employing search by extension and focus for solving the TSP but using different focus functions work together in a distributed environment using our TEAMWORK method. Several experts as well as referees have been written for DOTT, therefore we are able to combine many different approaches in cooperation with each other. Figure 6 shows a typical team run when using experts employing genetic algorithms only. The initial population is created by specialists; afterwards the experts improve the population. The referees judge the work of the experts and the supervisor sets up a new start population. The number of cycles depends on the desired goal.

As can be seen in experiments (see section 4), the genetic algorithms are the most powerful ones we have found so far. Especially by using a local optimization step after each genetic operation, the experts are able to find excellent solutions which are only about 0.5 % longer than the best solution. Moreover, with the TEAMWORK method, we can even find better solutions than any of the experts working alone.

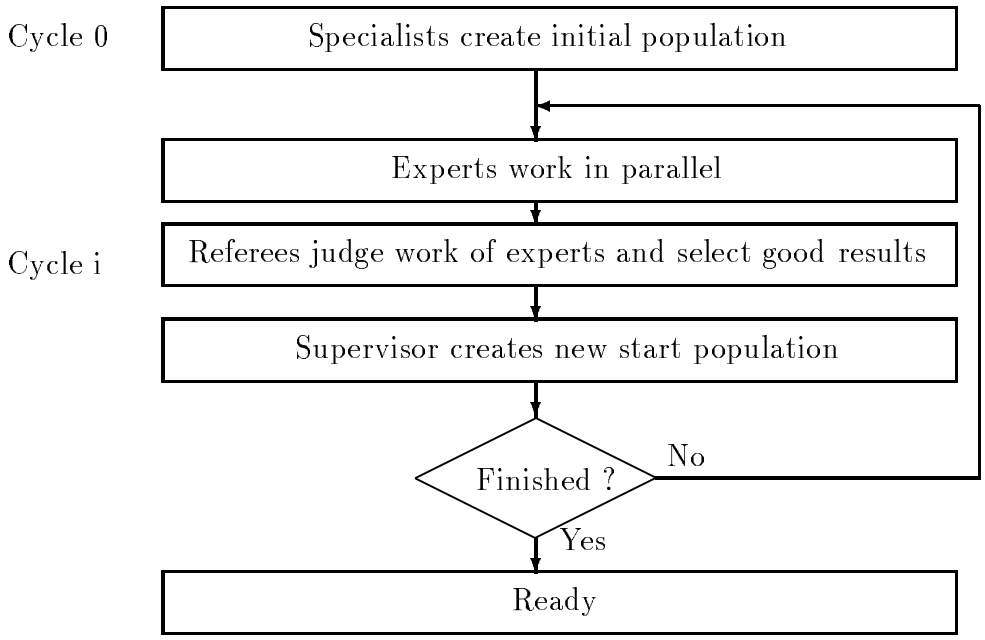

Figure 6: A typical team run

In the following, we will first present the Traveling Salesman Problem that we want to solve. After this we concentrate on the realizations of the different team components. We will follow the ordering given in Figure 6 and start with specialists and continue with experts, referees, and the supervisor.

3.1 The Traveling Salesman Problem

Given is a set of n cities along with their distances to each other. The Traveling Salesman Problem is finding the shortest round-trip, such that each city is visited exactly once. This problem can be seen in a geographical way, or more general from a theoretical point of view; several optimization problems can be transformed into Traveling Salesman Problems. We give in the following a more formal definition of the TSP.

Definition 3.1 (Traveling Salesman Problem)
Let $C = (c_{ij})$, $i, j \in \{1, \ldots, n\}$ *be a cost matrix.*

The problem of finding a permutation π *of* $\{1, \ldots, n\}$ *such that*

$\sum_{i=1}^{n} c_{\pi(i),\pi((i \bmod n)+1)}$ *is minimal*

is called the Traveling Salesman Problem.

The cost matrix represents the distances of the different cities to each other (c_{ij} is the cost for traveling from city number i to city number j). Note that for the general TSP,

it is not necessary that $c_{ij} = c_{ji}$ for all $i, j \in \{1, \ldots, n\}$. However, if this equation does hold, we speak of the symmetric TSP. Since it is the most common variant of the TSP, we consider only the symmetric TSP. For reference, see also [Re94] and [LLRS85].

3.2 Specialists

We will now explain the three most important specialists that are integrated in DOTT. Specialists are necessary for performing special tasks. For example, experts that use genetic algorithms need a specialist that creates an initial population (specialist FI). Other experts need some special additional information (specialist SORT). Furthermore, specialists can be used to help the supervisor control the optimization process and determine the end of a run (specialist LOWER).

3.2.1 Farthest Insertion

FI (Farthest Insertion) is a tour construction heuristic. All of the experts used in DOTT are tour improvement heuristics. Therefore it is necessary to construct an initial population of individuals first, using only the cost matrix as a basis. The actual algorithm works as follows (as described in [Jo90]):

1. Take one city as starting point for a "tour" consisting only of this city.
2. Pick the city that has the longest distance from the so-far generated tour, i.e. the city whose minimal distance to the cities of the tour is maximal. Add the city to the tour in the way that the cost for the insertion is minimal.
3. Repeat the second step until all the cities of the TSP have been added to the tour.

The idea of this algorithm is that by adding the farthest cities to the tour first, the hardest part is included first thus creating a rough "skeleton" which develops into a complete tour of a fairly good quality. The resulting tour differs, depending on which city is chosen to be the start city. Consequently there can be at most as many different tours as there are cities in the given TSP.

Farthest Insertion has shown to be a very good tour construction heuristic, especially compared to related heuristics (Nearest Insertion, Cheapest Insertion, Arbitrary Insertion, see [RSL74]). In experiments individuals created by FI were approximately 7 % longer than the optimal tour.

3.2.2 SORT

SORT is a specialist that processes information that is needed by some of the experts, e.g. the modified 3-Opt heuristic (see section 3.3.2). SORT creates sorted lists of distances to neighbor cities of the given TSP. Such a list for one city consists of the

indexes of the other cities of the TSP, sorted in ascending order. This way, it is easy to know the shortest edges starting from a given city. Such an ordered list is computed for each of the cities.

3.2.3 LOWER

LOWER calculates a lower bound of the TSP, i.e. a minimum length of the optimal tour. The algorithm is explained in [HK70]. The calculated bound is helpful for controlling the optimization process. It can be used to determine the quality of a tour; if a tour is created of the length of the lower bound, then this tour is optimal.

3.3 Experts

After setting up an initial population, it is necessary to improve it using experts. Although there are several different experts already integrated in DOTT, we achieved the best results with those experts employing genetic algorithms and using local optimization. Therefore we mainly concentrate on these experts. Nevertheless we would like to give some details about some of the other experts in DOTT first, since the ideas used in them will also be used in the experts of our experiments.

3.3.1 2-Opt

With this algorithm (see [Cr58]), two edges of a tour are chosen for a so-called 2-Opt-step. The edges are being removed from the tour and the resulting two fragments are rejoined in order to create a different, hopefully shorter tour. Note that for a 2-Opt-step there is only one possibility to join the two parts to a new tour if one wants to get a different one than the tour that was used for the optimization step in the first place. Of course the 2-Opt-step is only applied if the resulting tour is shorter than the original one. Figure 7 shows a 2-Opt step. The selected edges for performing 2-Opt as well as the ones inserted into the tour are marked by a symbol (*). Since we only deal with the symmetric TSP, the calculation of the new tour length is very easy by simply adding resp. subtracting the costs of the four edges (for the symmetric TSP, a reversed partial tour has the same cost as the original one).

If a tour cannot be optimized by any 2-Opt-step anymore, it is considered to be *locally optimal* with respect to 2-Opt. The same holds for related optimization heuristics like 3-Opt.

In order to reduce the complexity of the heuristic, we developed a slightly different 2-Opt heuristic. Our expert is able to reduce the number of optimization steps by considering only a part of the edges of a tour for a 2-Opt step, namely those which are very likely to bring an improvement of the tour. This way the run time of the expert is improved while the quality of the results differ only a little bit compared to the standard heuristic.

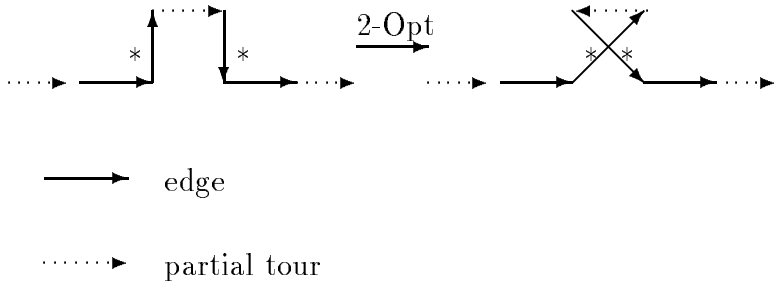

Figure 7: A 2-Opt step

There are two parameters for controlling this heuristic. First, the parameter *maxnum* sets a maximum for the number of 2-Opt steps applied to one tour, in order to prevent the system from operating only on a few tours while neglecting the others. Furthermore the parameter *depth* reduces the number of considered edges for 2-Opt; the idea is that not all of the edges are tested for rejoining the tour fragments, but only those that very likely lead to an improvement of the tour. Since the resulting tour should be shorter than the original one, it makes sense to add one of the shortest edges to the tour in a 2-Opt step. Let (a_i, a_j) be the shortest edge starting from city a_i. This is determined with the help of the sorted list of neighbor cities for city a_i, as calculated by the specialist SORT. If this edge is not already in the tour, there are only two possibilities for applying a 2-Opt step if this new edge should be present in the resulting tour. Consequently, two different tours can be generated this way. Figure 8 shows these two possibilities. The chosen edges as well as the inserted ones are marked by a symbol (*).

It has been shown in experiments, that by reducing the number of considered edges for applying a 2-Opt step this way, 2-Opt runs much faster without losing much of the quality of the generated tours.

3.3.2 3-Opt

Similar to 2-Opt, this heuristic breaks a tour in different parts and rejoins them in the way that a different tour is created (see [Lin65]). With 3-Opt, three edges are chosen; there are eight possibilities of rejoining the resulting three parts to a new tour. Again, this is only done if a shorter tour is achieved this way. Note that the 2-Opt-heuristic is included in 3-Opt; a 2-Opt step may be done by reusing one of the chosen edges in a 3-Opt step. Figure 9 shows two possible 3-Opt steps.

3-Opt has a much higher complexity than 2-Opt. The resulting tours are in average better than the ones generated by 2-Opt. It is necessary to decide whether it is more important to get results of a good quality or a low processing time.

In DOTT, our expert uses a modified 3-Opt, which is the obvious progression of the modified 2-Opt described in section 3.3.1. Again, only a part of the edges are taken into consideration for applying 3-Opt steps (those that are very likely to lead to an

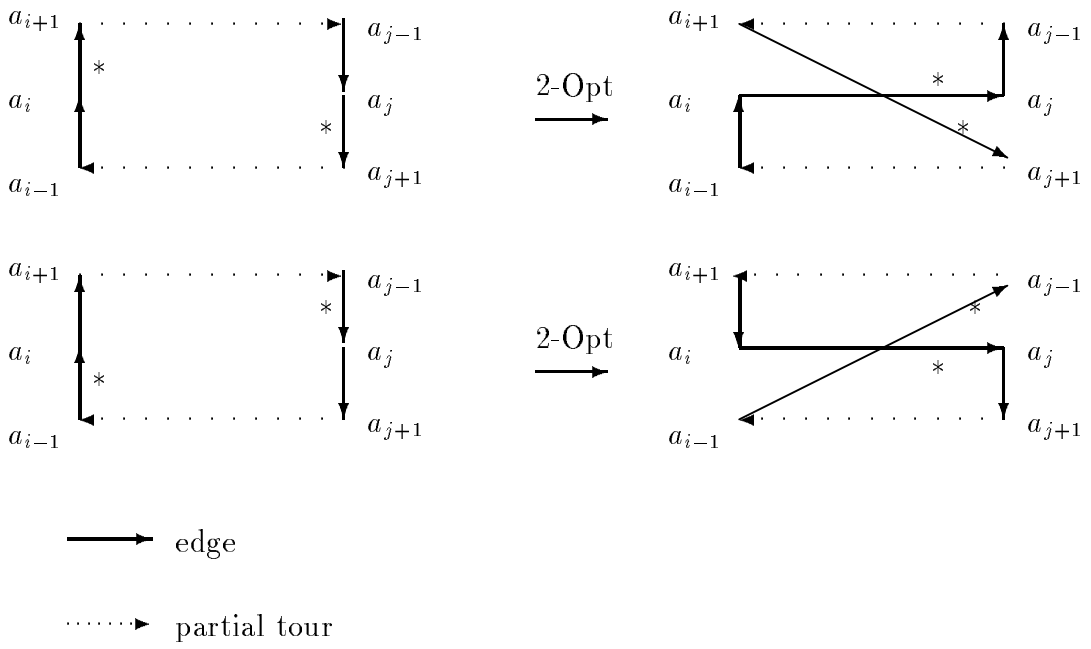

Figure 8: Our modified 2-Opt

improvement of the tour). The idea is that if a new edge is used, it should be as short as possible. Starting from city a_i, this edge is determined with the help of the sorted list of neighbor cities for a_i. If the edge (a_i, a_j) resp. (a_j, a_i) should be found in the resulting tour, there are two different possibilities (with several different values for k) for performing a 3-Opt step (Figure 10).

With this modified 3-Opt heuristic, the run time improves very much, since not all of the possible 3-Opt steps are actually applied, while the quality of the results is very close to the one achieved by the original 3-Opt heuristic.

3.3.3 Edge Recombination Crossover

With this genetic algorithm (see [Mi92]), two parent individuals are chosen for performing a crossover step. The parents are chosen with respect to the fitness function and a random number generator. The essential idea of the heuristic is that those edges should be found in the child that are in both of the parents or at least in one parent. In order to make sure that the child is indeed a correct individual, i.e. a TSP tour, it is necessary to also use edges that are not found in any of the parents; if an edge of a parent cannot be used without getting an incorrect individual, we have an *edge failure*. An edge failure is treated by using a different edge not found in any of the parents. Consequently, ER Crossover is not a simple crossover, but actually a combination of crossover and mutation. Still, since it is the essential idea, it makes sense to call it crossover. In more detail, the algorithm works as follows.

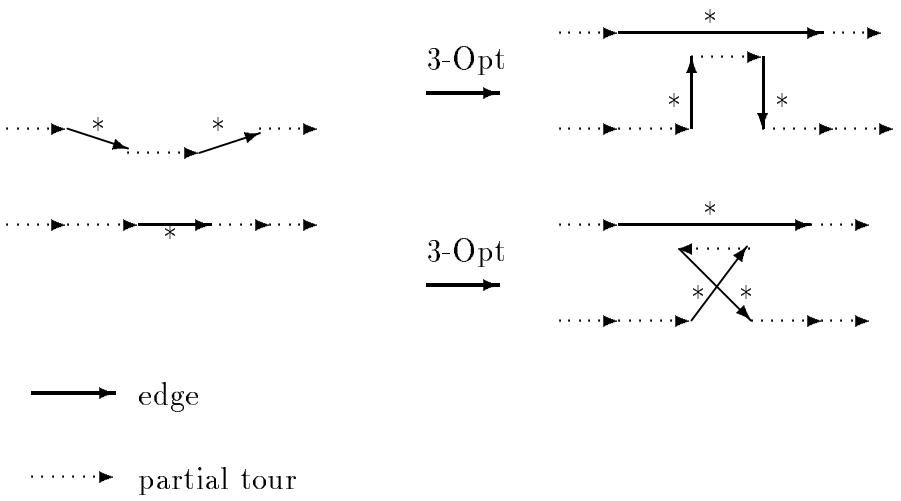

Figure 9: Possible 3-Opt steps

1. Choose two parent tours for performing ER Crossover (this is controlled by the parameters p and $pcorr$).
2. Pick a start city (controlled by the parameter $vfirst$)
3. Choose the next edge for inserting into the resulting tour. If it is possible to use an edge (starting from the active city) which can be found in one of the parents, choose it (controlled by parameter $vnext$). Else select a new edge which cannot be found in any of the parents (controlled by parameter $vnone$).
4. If the resulting tour is not complete yet, go to step 3

For the selection of the parents, a random number generator is used, such that a probability distribution is accomplished. In more detail, p is the probability that the first tour (i.e. the shortest one) is chosen as parent; the probability for choosing the second tour (i.e. the second-shortest one) is $p * pcorr$; for the third tour $p * pcorr^2$ and so on. If the random number (adjusted to the interval [0,1]) is higher than the actual probability, then the individual is chosen. Our expert is able to use different probability distributions this way for the selection of the parents. For example one may use primarily short tours by setting p:=50, $pcorr$:=100.

If two parents have been selected, the crossover according to the algorithm can be controlled by the following parameters.

vfirst This parameter determines the heuristic for choosing the start city. It may be assigned the following values:

FIRST_RAND: Select a city with equal probability for each city.

FIRST_MIN_ALL: Select a city with equal probability only from the set of those cities that have the most common edges in both parents.

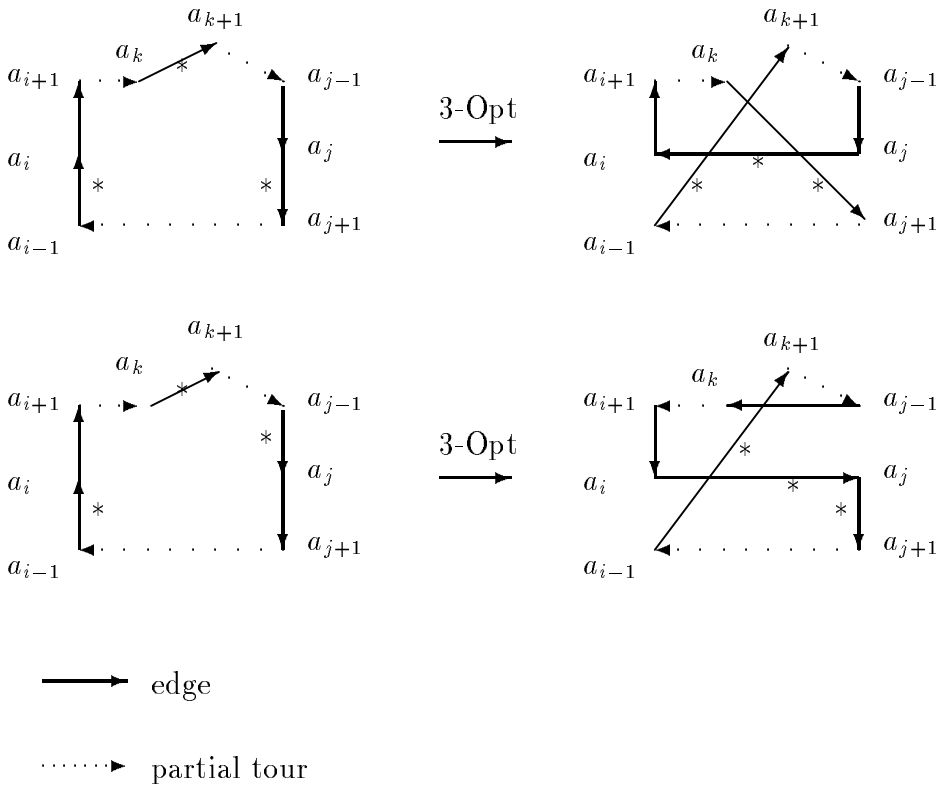

Figure 10: Our modified 3-Opt expert

FIRST_MIN_5: Randomly pick five different cities. Choose the city out of the five picked ones that has the most edges in common for both parents (if there is more than one city of the same quality, choose the first one).

vnext This parameter determines the strategy for choosing the next city while constructing the child tour. It may have one of the following settings:

NEXT_RAND: Pick one of the maximal four possible edges of the parents with equal probability. Leave out those edges that lead to cities that have already been used for the child tour.

NEXT_MIN_USED: For each of the maximal four cities determine the number of edges that have not been used so far. Out of those cities that lead to the minimal number, choose one with equal probability.

vnone This parameter determines the strategy for choosing the next edge if it is not possible to use an edge of the parents.

NONE_RAND: Pick a city with equal probability out of the set of those cities that have not been used so far.

NONE_SORTED: Pick the nearest possible city.

3.3.4 Edge Recombination Crossover with Local Optimization 3-Opt (ER_LO3)

One of the main disadvantages of genetic algorithms is that they tend to be quite slow. A different approach for an expert is the combination of genetic algorithms with effective tour improvement heuristics (see section 2.1). In DOTT, there are four different experts that use this approach (ER_LO2, ER_LO3, MUT4_LO2, MUT4_LO3). As genetic operators we chose Edge Recombination Crossover and 4-Opt-Mutation, as tour improvement heuristics we used 2-Opt and 3-Opt.

ER_LO3 randomly chooses two individuals as parents, performs an Edge Recombination Crossover step on them and improves the resulting child with the (modified) 3-Opt-heuristic. The main cost for this expert lies in the 3-Opt optimization. Therefore ER_LO3 produces only few individuals compared to the experts that use 2-Opt as local optimization. The experts that use 3-Opt approximately need the same running time for producing a new tour; the same holds for the 2-Opt-experts.

ER_LO3 uses the following parameters. First the parameters which can be found in the ER Crossover expert p, $pcorr$, $vfirst$, $vnext$, $vnone$ are present again. The parameters for the 3-Opt optimization are $maxnum$ and $depth$. Moreover, there can be more than one generation for this expert; the permitted time for one generation is determined by the parameter $gentime$. After this time, the expert substitutes the active generation by those individuals that have been created. Note that our expert strictly separates generations; this means that newly created individuals cannot be immediately used for progressing in the search process. Moreover individuals (except for the best one) are not taken over to the next generation, except if they are generated again by the expert. The same holds for the other experts ER_LO2, MUT4_LO3 and MUT4_LO2. Consequently, we have an explicit selection rule that deletes the old individuals after the generation time has elapsed.

Here is the algorithm in more detail.

For each generation do:

1. Randomly pick two parent tours from the population using the parameters p and $pcorr$
2. Perform ER Crossover on these two tours (parameters $vfirst$, $vnext$, $vnone$)
3. Local optimization: use 3-Opt on the resulting tour (parameters $maxnum$, $depth$)
4. Add the resulting, optimized tour to the new generation
5. If the generation time is not over (parameter $gentime$), go to step 1

If the generation time is over, the best tour of the original population is added to the new population, if it hasn't been created anyway. Then the next generation is used as the new population while the old generation is erased.

In experiments, the best tours produced by this expert were in average 0.2 % longer than the optimal tour.

3.3.5 Edge Recombination Crossover with Local Optimization 2-Opt (ER_LO2)

The only difference between this expert and ER_LO3 is the local optimization method used. Instead of 3-Opt, this expert uses 2-Opt. The main idea of this heuristic is that this way much more individuals are created and therefore a larger variety of tours is achieved in the population. Indeed, as can be seen in experiments, this expert is much faster in producing new tours than the ones that use 3-Opt as local optimization; results of a comparable quality can be found faster with it. The parameters for this expert are the same as for ER_LO3: p, $pcorr$, $vfirst$, $vnext$, $vnone$, $maxnum$, $depth$ and $gentime$. The parameters $maxnum$ and $depth$ are used for the 2-Opt local optimization.

The best results we got from this expert in experiments were approximately 0.63 % longer than the optimal tour.

3.3.6 4-Opt-Mutation with Local Optimization 3-Opt (MUT4_LO3)

A simple mutation of a TSP tour one can think of is exchanging two cities of the tour. For this, four edges have to be substituted by four other ones. 4-Opt is a more general heuristic for performing this; it is not a mutation in the first place, but the next consequence of the tour improvement heuristics 2-Opt and 3-Opt. Four edges are chosen for breaking an existing tour into four parts. Afterwards these four partial tours are rejoined to a new tour. Normally, since we have a tour improvement heuristic, the resulting tour should be shorter than the original one. With this expert, however, this is not necessary. We only use one special 4-Opt step for a modification (mutation) of the tour; afterwards the local optimization heuristic 3-Opt improves the resulting tour. The algorithm is very close to a heuristic used in [Jo90], with the differences that we use 3-Opt for local optimization instead of the Lin-Kernighan heuristic and we have a population of tours and not only one.

After the four edges are (randomly) chosen for the mutation step, the actual 4-Opt-step is deterministic; we used a special one instead of trying out all of the possibilities. In Figure 11, the 4-Opt step that we used is illustrated.

Parameters for this expert are p, $pcorr$ (the selection of the tour for mutation is done in the same way as for ER_LO3), $maxnum$, $depth$ (for the local optimization 3-Opt) and $gentime$ (for the generation time; see ER_LO3).

The algorithm works as follows.

For each generation do:

1. Randomly choose one tour from the population using the parameters p and $pcorr$
2. Perform the 4-Opt mutation step as illustrated in Figure 11 on this tour (the edges are chosen randomly)
3. Local optimization: use 3-Opt on the resulting tour (parameters $maxnum$, $depth$)

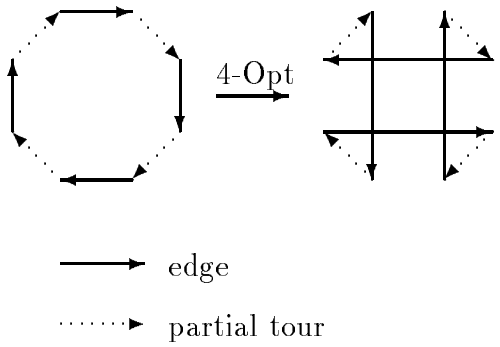

Figure 11: The 4-Opt-step for our 4-Opt-mutation experts

4. Add the resulting, optimized tour to the new generation

5. If the generation time is not over (parameter *gentime*), go to step 1

The substitution of the generation, if the generation time is over, is done in the same way as for ER_LO3.

In experiments we found that the best results of this expert were about 0.28 % longer than the optimal tour.

The time for generating a new tour of the expert is very similar to the one of ER_LO3. Therefore these two experts are easily able to work together in a team.

3.3.7 4-Opt-Mutation with Local Optimization 2-Opt (MUT4_LO2)

This expert works in the same way as MUT4_LO3. The only difference is that for local optimization the 2-Opt heuristic is used instead of 3-Opt. The parameters for this expert are the same ones as for MUT4_LO3: p, $pcorr$, $maxnum$, $depth$ and $gentime$ (generation time). It is obvious that this expert can be used very well together with ER_LO2, since they have comparable running times for producing new individuals.

In experiments, the best tours generated by this expert were approximately 0.33 % longer than the optimal tour. Again this shows that it seems to be a successful approach to use a local optimization heuristic of a lower complexity. Good results are found much faster with MUT4_LO2; moreover they are of about the same quality as the ones generated by MUT4_LO3 or ER_LO3.

3.4 Referees

As described in section 2.3, referees in TEAMWORK have two different tasks: first the judgment of the overall work of an expert ($meas_R$); second the selection of good results in order to send them to the supervisor ($selres_R$). In DOTT, there are several different

referees with different $selres_R$ and $meas_R$ functions. We will describe one version of each of the functions in more detail.

3.4.1 The $selres_R$ Function DIVERSITY

This function is used for selecting good results and sending them to the supervisor in order to set up a new start population for the next cycle of TEAMWORK. It is not easy to decide which of the tours have to be considered as "good", because it may be necessary to select longer tours as well as short ones in order to achieve a large variety of different tours. This is necessary for being able to escape local optima. Figure 12 gives an illustration of this problem. For example, a $selres_R$ function that only takes into account the quality (i.e. the length) of a tour, will choose a tour which is located at x1 instead of one that is located at x2. During the further optimization process, the tour located at x1 will develop towards the local optimum at x3, while the other tour could have developed towards x4 and afterwards towards the global optimum x5. Therefore a simple function as described above may produce good progress in the first part of the optimization process, but also has the tendency of entering local optima. Our $selres_R$ function is able to cope with this problem by choosing individuals not only according to their quality (length), but also according to their similarity towards other tours (we compare it with the shortest tour of the population).

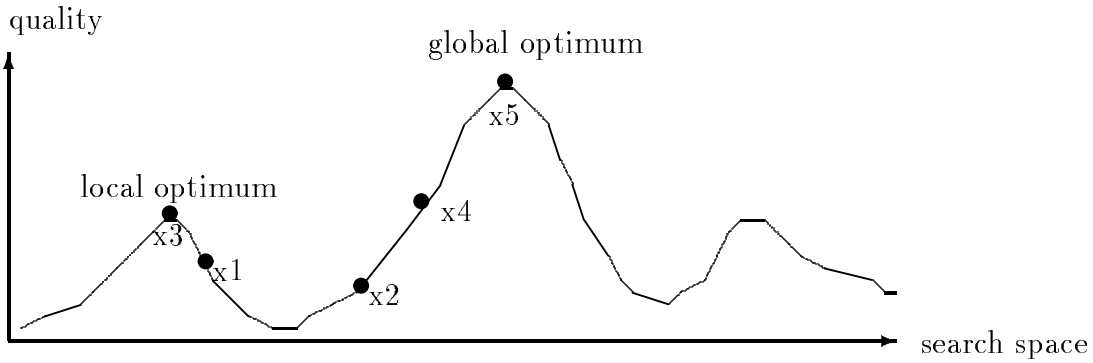

Figure 12: Avoiding local optima

It is not easy to define what "similarity" means for TSP tours; we decided the most useful approach is to count the number of equivalent edges of two tours in order to decide how similar they are to each other.

Our $selres_R$ function uses two different parameters:

p_{length} determines the importance of the length of a tour; the larger this parameter, the more relevance is set on it.

$p_{diversity}$, on the other hand, determines the importance of the similarity of a tour compared to the shortest tour; the larger this parameter, the more relevance is set on it.

For the two parameters, we suggest per cent or per thousand values. Each tour π is given a relevance value using the following formula:

$$
\begin{aligned}
relevance(\pi) &= p_{length} * f_{length}(\pi) + p_{diversity} * f_{diversity}(\pi) \\
&\text{with} \\
f_{length}(\pi) &= \frac{\text{length of the shortest tour}}{\text{length of } \pi} \\
f_{diversity}(\pi) &= \frac{\text{number of non common edges of } \pi \text{ with shortest tour}}{\text{number of cities of the given TSP}}
\end{aligned}
$$

Those tours that have the highest relevance values are selected for sending to the supervisor. This way it is possible to select mainly short tours, or on the other hand mostly those tours that are very different from the shortest one. The only exception is the shortest generated tour; it will be sent to the supervisor in any case.

3.4.2 The $meas_R$ Function LENGTH

For this function, the length as well as the number of the produced tours are taken into account in order to judge the work of an expert. It can be controlled by a number of parameters, which decide what aspects of the results should be primarily regarded.

num: this parameter gives the (maximum) number of relevant tours for the parameters $p_{avg\_shortest}$, $p_{avg\_longest}$ and $p_{avg\_unsimilar}$

$p_{shortest}$: the larger this parameter, the more the length of the shortest generated tour is taken into account

p_{avg_all}: the larger this parameter, the more the average length of all tours is taken into account

$p_{avg\_shortest}$: the larger this parameter, the more the average length of the num shortest tours is taken into account

$p_{avg\_longest}$: the larger this parameter, the more the average length of the num longest tours is taken into account

$p_{avg\_unsimilar}$: the larger this parameter, the more the average length of the num most unsimilar tours (compared to the shortest tour) is taken into account

p_{gen}: the larger this parameter, the more the number of generated tours is taken into account (quantity)

A measure for the work of the expert X is calculated with the following formula:

$$
\begin{aligned}
success(X) &= p_{shortest} * f_{shortest}(X) + p_{avg\_all} * f_{avg\_all}(X) + \\
&\quad p_{avg\_shortest} * f_{avg\_shortest}(X) + p_{avg\_longest} * f_{avg\_longest}(X) + \\
&\quad p_{avg\_unsimilar} * f_{avg\_unsimilar}(X) + p_{gen} * f_{gen}(X)
\end{aligned}
$$

with

$$
\begin{aligned}
f_{shortest}(X) &= \frac{\text{normalization}}{\text{length of the shortest tour of X}} \\
f_{avg\_all}(X) &= \frac{\text{normalization}}{\text{average length of all tours of X}} \\
f_{avg\_shortest}(X) &= \frac{\text{normalization}}{\text{average length of the } num \text{ shortest tours of X}} \\
f_{avg\_longest}(X) &= \frac{\text{normalization}}{\text{average length of the } num \text{ longest tours of X}} \\
f_{avg\_unsimilar}(X) &= \frac{\text{normalization}}{\text{av. length of } num \text{ tours of X most unsimilar to shortest one}} \\
f_{gen}(X) &= \frac{\text{number of tours produced by X}}{\text{number of cities of the TSP}}
\end{aligned}
$$

The following unequations hold for the different factors:

$$
\begin{aligned}
0 &< f_{shortest} < \infty, \text{in general } 1 \leq f_{shortest} < 1.1 \\
0 &< f_{avg\_all} < \infty, \text{in general } 1 < f_{avg\_all} < 1 \\
0 &< f_{avg\_shortest} < \infty, \text{in general } 1 < f_{avg\_shortest} < 1 \\
0 &< f_{avg\_longest} < \infty, \text{in general } 1 < f_{avg\_longest} < 1 \\
0 &< f_{avg\_unsimilar} < \infty, \text{in general } 1 < f_{avg\_unsimilar} < 1 \\
0 &\leq f_{gen} < \infty, \text{in general } 1 \leq f_{gen} < 2
\end{aligned}
$$

The parameters amplify these factors, therefore reasonable settings for the parameters are per cent or per thousand values.

For a normalization factor, we used the length of the shortest tour of the start population (or the number of cities of the TSP, if the start population is empty). This way we can compare the new results with the shortest tour as it was given to the expert. If, for example, the expert generates a new shortest tour, then the value of $f_{shortest}$ will be higher than 1; the higher this value, the better the new generated shortest tour. The other factors are to be read in the same way: the higher the values, the better the work of the expert in regard to this attribute.

The larger the success value of an expert, the better its work is considered; consequently the winner is the expert with the largest success value.

3.5 The Supervisor

The supervisor is the coordinator of the whole optimization process. It composes a new start population for each new cycle based on the selection of good tours by the referees (*Comp*). The actual team consisting of experts, specialists and referees is chosen by it (*Sel*). Finally it decides how much time should be given to the experts resp. specialists for the next computation cycle (*Time*). In DOTT, the selection of the right experts and specialists as well as the time is achieved by a control file which is created before an actual optimization run. The winning expert is appointed by the supervisor based on the judgments of the referees. The composition of a new start state works in the way that all the results of the winning expert are used. Additionally the best results of the other experts are added to the new start population. To put it into a more formal way, our supervisor is characterized by the following functions:

$$
\begin{aligned}
Comp((x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)) &= x_k \cup \bigcup_{i=1,\ldots,n}^{i \neq k} y_i, \text{if } z_k = \max\{z_i\} \\
Sel_i(x, y) &= 1 \text{ iff } x \text{ is expert or specialist and } y \text{ is referee} \\
&\quad \text{for cycle } i+1 \text{ according to control file.} \\
Time_i(x, (y_1, z_1), \ldots, (y_n, z_n)) &= \text{time read from control file for cycle } i+1
\end{aligned}
$$

In addition to determining the length of the next working period the supervisor also determines the end of a team run (which means that all future working periods have length 0). In DOTT, a team run is stopped if either a given time limit is reached, or a given number of cycles, or if the best tour found has a length which is a given percentage higher than the result of specialist LOWER.

Even with this somewhat basic supervisor very good results could be achieved in experiments, as can be seen in the following section.

4 Experiments

In this section we present results of experiments that have been made with the system DOTT, in order to examine whether the ideas explained in the previous sections show good results in reality. For this, we first take a look at experiments with experts (or specialists) working alone; we call these *single runs*. Afterwards, we present results of experiments in which experts (and specialists) work in cooperation with each other using TEAMWORK, the so-called *team runs*.

DOTT is written in ANSI-C. It is compiled on the operating system SUN OS 4.1 using the GNU C compiler. All of the experiments have been made on one resp. two SUN SPARCStations 10. For TSP instances, we used the Traveling Salesman Problem library TSPLIB version 1.2 (see [Re91]). Since our experts use a random number generator, it is generally not possible to reproduce a single run, or even a team run. A

run depends on the load of a machine and also of the communication network, since we have a distributed system. Still, the experiments clearly show tendencies, since we tested several different TSP instances as well as parameter combinations.

A test run, no matter if it is a single or a team run, is divided into two parts. The first part is the creation of an initial population; we used the specialist FI (Farthest Insertion) for this task. In the second part, experts improve the population; we achieved the best results with the experts employing genetic algorithms and using local optimization, namely ER_LO3, ER_LO2, MUT4_LO3 and MUT4_LO2. These four experts need the lists of sorted neighbor cities; therefore it is necessary to use the specialist SORT first. For the single runs, this is done immediately after the specialist FI is finished. For the team runs, FI and SORT work in parallel.

The time values are written in the format **hours minutes:seconds.centiseconds**. They indicate the time when first a tour of the indicated length was created. If the tour is an optimal one, it is marked by an asterisk (*).

4.1 Single Runs

The attributes of our selected TSP instances as well as the results of the tour construction cycle are given in Table 1. The first column gives the name of the TSP instance; the number included in the name indicates the number of cities of this instance. Column 2 gives the optimal tour length of the instance as we know it from literature. Columns 3 and 4 give the time we set for the tour construction cycle and the number of generated tours by FI. Finally, the length of the best created tour by FI is given in column 5 along with the information of how much longer it is than the optimal tour. Note that the results for FI can differ, as explained before. However, the differences are only very minor and have almost no influence on the further optimization process.

TSP	optimal length	time for FI	# tours gen.	shortest tour (quality)
kroB100	22 141	1.20	91	22 693 (2.49%)
gr202	40 160	10.00	197	41 875 (4.27%)
lin318	42 029	1:00.00	304	44 262 (5.31%)
pcb442	50 778	2:30.00	375	55 373 (9.05%)
att532	27 686	5:00.00	516	29 198 (5.46%)
ali535	203 310	3:00.00	336	214 228 (5.89%)
d657	48 912	6:00.00	386	52 211 (6.74%)
gr666	294 358	11:40.00	614	318 627 (8.24%)
u724	41 910	13:20.00	709	45 120 (7.66%)
rat783	8 806	16:40.00	625	9 587 (8.87%)
dsj1000	18 659 688	33:20.00	816	20 044 307 (7.42%)
u1060	224 094	17:00.00	391	243 435 (8.63%)

Table 1: TSP instances with tour construction cycle (FI)

Obviously the best tours constructed by FI are approximately 6.67 % longer than the optimal tour. Let us see next, which improvements can be achieved by our four experts

in single runs. Note that the construction cycle of the initial population is the same for all our test runs.

4.1.1 MUT4_LO2

Table 2 gives the results for the expert MUT4_LO2. Column 1 gives the name of the TSP instance, column 2 gives the the time applied for creating one generation in seconds. The third column gives the length of the best found tour along with the information of how much longer it is than the optimal tour; column 4 gives the creation time of this tour. Finally, the last column gives the total time of the test run.

For the parameters, we tested several different values for the generation time *gentime* as well as for *depth*. The table shows the results of the best found combination. By setting p=50 and *pcorr*=100, we preferred the shortest tours in the selection. *maxnum* is assigned a value which is normally not reached; we found out in experiments, that a tour is locally optimal after just a few optimization steps. Therefore it is not necessary to set a low bound in this case. As for the parameter *depth*, in some cases a lower value produced better results, in other cases it was the opposite way. We tried different possibilities; the best result is given in the table.

To sum it up, the parameters were assigned the following values:

$$\begin{aligned} maxnum &= 200 \\ depth &= 3 \text{ resp. } 5 \\ p &= 50 \\ pcorr &= 100 \\ gentime &= \text{generation time; differs for different instances} \end{aligned}$$

TSP	gen. time	best result (quality)	time	maximum time
kroB100	0.2 s	*22 141 (0.00%)	2.46	21.30
gr202	0.5 s	*40 160 (0.00%)	20.46	1:50.20
lin318	1.5 s	42 203 (0.41%)	2:24.83	2:27.78
pcb442	2.5 s	50 991 (0.42%)	5:20.53	15:01.00
att532	3 s	27 731 (0.16%)	21:39.44	25:01.00
ali535	12 s	203 011 (0.35%)	15:26.52	22:49.25
d657	16 s	49084 (0.35%)	31:02.50	39:05.48
gr666	4 s	295 731 (0.47%)	31:26.12	45:01.50
u724	20 s	42 080 (0.41%)	1h 47:23.76	1h 47:23.76
rat783	20 s	8 816 (0.11%)	1h 47:50.97	1h 57:44.88
dsj1000	24 s	18 749 543 (0.48%)	2h 33:37.87	2h 53:24.85
u1060	6 s	225 995 (0.85%)	1h 11:49.14	1h 27:00.54

Table 2: MUT4_LO2: results of single runs

As can be seen from table 2, the best results we found for MUT4_LO2 are approximately 0.33 % longer than the optimal tour. This is an enormous improvement in comparison

to the results produced by FI. For two instances, even the optimal tour could be found; the other results are all less than 1 % longer than the optimal tour. Although the run time for the expert is less than 3 hours for a 1000 city problem, a very good solution could be found with MUT4_LO2 (instance dsj1000, only 0.48% longer than the optimal tour).

4.1.2 MUT4_LO3

The results for the expert MUT4_LO3 can be found in Table 3. The parameters we used for it are very similar to the ones for MUT4_LO2; the generation time needs to be much higher though, since the local optimization (3-Opt) is much more complex than 2-Opt used by MUT4_LO2. We used the following settings:

$$
\begin{aligned}
maxnum &= 200 \\
depth &= 3 \text{ resp. } 5 \\
p &= 50 \\
pcorr &= 100 \\
gentime &= \text{generation time; differs for different instances}
\end{aligned}
$$

TSP	gen. time	best result (quality)	time	maximum time
kroB100	1s	*22 141 (0.00%)	11.97	15.90
gr202	5 s	*40 160 (0.00%)	1:41.49	6:15.85
lin318	20 s	42 152 (0.29%)	7:48.50	8:38.29
pcb442	40 s	50 814 (0.07%)	41:53.86	47:48.80
att532	75 s	27 743 (0.21%)	45:38.73	1h 00:21.32
ali535	75 s	202 529 (0.11%)	42:21.14	59:16.51
d657	240 s	48 992 (0.16%)	1h 58:40.57	2h 42:02.11
gr666	480 s	294 831 (0.16%)	2h 21:00.76	2h 33:52.87
u724	250 s	42 001 (0.22%)	3h 19:20.56	3h 36:23.15
rat783	250 s	8 862 (0.64%)	3h 39:47.31	3h 45:03.26
dsj1000	1800 s	18 857 869 (1.06%)	6h 30:40.40	6h 33:26.99
u1060	450 s	225 194 (0.49%)	6h 16:16.86	6h 24:35.58

Table 3: MUT4_LO3: results of single runs

As can be seen from Table 3, the best tours generated by MUT4_LO3 are in average 0.28 % longer than the optimal tour. As expected, this expert produces better results than MUT4_LO2 due to its more powerful local optimization heuristic. For two instances, the optimal tour could be found; most of the other results are less than 0.3 % longer than the optimal solution. However, the run time for MUT4_LO3 is much higher than for MUT4_LO2, again because of its more complex local optimization method (approximately about five times higher). Therefore, since the results are in average not

very much better than the ones obtained by MUT4_LO2, the approach of using a less complex local optimization method has proved to be successful. For some instances (att532, rat783, dsj1000), MUT4_LO2 even found a better tour than MUT4_LO3.

We now examine if similar results can be found with our other two genetic experts, ER_LO3 and ER_LO2.

4.1.3 ER_LO2

The values for the parameters for ER_LO2 are very close to the ones used for MUT4_LO2. $maxnum$ is set to 200; for different generation times $gentime$ as well as for $depth$ values we used the same combinations as for MUT4_LO2, since they need similar time for generating a new tour. Again the best results can be found in table 4.

The parameters are set to the following values:

$$\begin{aligned} maxnum &= 200 \\ depth &= 3 \text{ resp. } 5 \\ vfirst &= \text{FIRST\_MIN\_5} \\ vnext &= \text{NEXT\_MIN\_USED} \\ vnone &= \text{NONE\_SORTED} \\ p &= 50 \\ pcorr &= 100 \\ gentime &= \text{generation time; differs for different instances} \end{aligned}$$

As can be seen from Table 4, the best found tours are approximately 0.63 % longer than the optimal ones. The optimal tour could be found for one instance (gr202). The results are slightly worse than the once obtained by MUT4_LO2. Still, a better tour could be found for instance u1060 by ER_LO2. This shows, that indeed different experts have different capabilities in regard to TSP instances. One expert may be good on a particular instance, while it may be worse on another one. Our TEAMWORK method helps to combine the advantages of these different capabilities.

4.1.4 ER_LO3

For the last expert we tested, ER_LO3, we used similar parameters to ER_LO2. $vfirst$, $vnext$ and $vnone$ are assigned the same values. The different generation times $gentime$ as well as the different values for $depth$ are the same ones as for MUT4_LO3.

Consequently the parameters for ER_LO3 are assigned the following values:

$$maxnum = 200$$

TSP	gen. time	best result (quality)	time	maximum time
kroB100	0.6 s	22 199 (0.26%)	3.24	1:01.30
gr202	2 s	*40 160 (0.00%)	29.17	6:50.20
lin318	2 s	42 346 (0.75%)	2:12.71	2:24.67
pcb442	2.5 s	51 126 (0.69%)	7:07.25	15:01.00
att532	12 s	27 785 (0.36%)	17:48.02	45:01.00
ali535	12 s	204 411 (1.04%)	5:25.36	23:01.21
d657	4 s	49 399 (1.00%)	7:18.31	39:18.05
gr666	16 s	296 276 (0.65%)	35:36.48	1h 18:21.50
u724	20 s	42 227 (0.76%)	22:43.64	1h 53:23.61
rat783	20 s	8 847 (0.47%)	1h 12:24.64	1h 56:44.02
dsj1000	24 s	18 803 744 (0.77%)	2h 16:15.65	2h 53:26.05
u1060	12 s	225 917 (0.81%)	1h 01:40.47	1h 27:04.22

Table 4: ER_LO2: results of single runs

$$
\begin{aligned}
depth &= 3 \text{ resp. } 5 \\
vfirst &= \text{FIRST\_MIN\_5} \\
vnext &= \text{NEXT\_MIN\_USED} \\
vnone &= \text{NONE\_SORTED} \\
p &= 50 \\
pcorr &= 100 \\
gentime &= \text{generation time; differs for different instances}
\end{aligned}
$$

TSP	gen. time	best result (quality)	time	maximum time
kroB100	4 s	*22 141 (0.00%)	19.29	1:01.30
gr202	20 s	*40 160 (0.00%)	1:52.83	8:30.20
lin318	10 s	42 071 (0.10%)	2:35.06	9:01.09
pcb442	160 s	50 795 (0.03%)	1h 29:38.14	1h 35:51.00
att532	75 s	27 704 (0.07%)	41:40.81	1h 01:16.00
ali535	150 s	202 978 (0.33%)	18:22.35	58:01.77
d657	480 s	48 975 (0.13%)	2h 23:30.94	2h 38:02.63
gr666	480 s	294 700 (0.12%)	2h 38:57.04	2h 43:41.50
u724	250 s	42 061 (0.36%)	3h 22:31.31	3h 36:32.96
rat783	250 s	8 846 (0.45%)	2h 43:47.24	3h 40:15.22
dsj1000	450 s	18 751 750 (0.49%)	6h 28:45.45	6h 33:25.70
u1060	450 s	224 909 (0.36%)	6h 10:46.95	6h 24:36.06

Table 5: ER_LO3: results of single runs

The best results of ER_LO3, according to Table 5, are in average 0.2 % longer than the optimal tour. Again the usage of a more powerful local optimization shows better results than a less complex one. The optimal tour could be found for two different instances (kroB100, gr202). Most of the other results are less than 0.37 % longer than

the optimal solution. However, the run time for ER_LO3 is much higher than for ER_LO2. It is necessary to choose between a better quality of the results and a lower run time. Still the quality of the results of ER_LO3 and ER_LO2 are quite close to each other, therefore the usage of a less complex local optimization method again has proved to be successful.

Of our four different experts, the best results could be found with ER_LO3 (0.2 % longer than the optimal tour). MUT4_LO3 produced results that are 0.28 % longer than the optimal tour, for MUT4_LO2 this value is 0.33 %. The worst results were found by ER_LO2, but even this expert achieved results of a rather good quality (0.63 %).

Since our four experts achieved results that are approximately 0.36 % longer than the optimal tour, they are already very good working alone. The results are very close to the optimal tour. Compared to our other experts realized in DOTT, these experts are by far the best ones. Therefore it is quite difficult to achieve even better results by using a distributed environment. In the next subsection, we examine the results obtained by our two different teams.

4.2 Team Runs

Since the time for generating a new individual primarily depends on the local optimization method used, it makes sense to let those experts cooperate in a team that use the same local optimization heuristic. Therefore, we used two different fixed teams in our test runs: MUT4_LO2 & ER_LO2 and MUT4_LO3 & ER_LO3. We present the results obtained by the 2-Opt team first.

4.2.1 2-Opt Team

As parameter values for this team, we used the same ones as those of the most successful single run of the two experts. Furthermore, we tested several different combinations of parameter values for the referee. In experiments, we found a parameter setting that shows good average results for almost all of the different TSP instances (see [Sch96]). We now give these settings for the 2-Opt team (the referee consists of the functions DIVERSITY and LENGTH).

$$
\begin{aligned}
&\text{DIVERSITY :} \\
&\qquad p_{diversity} = 75 \\
&\qquad p_{length} = 25 \\
&\text{LENGTH :} \\
&\qquad num = \text{about 10\% of the number of generated tours} \\
&\qquad p_{shortest} = 0 \\
&\qquad p_{avg\_all} = 0 \\
&\qquad p_{avg\_shortest} = 0
\end{aligned}
$$

$$
\begin{aligned}
p_{avg\_longest} &= 0 \\
p_{avg\_unsimilar} &= 0 \\
p_{gen} &= 1000
\end{aligned}
$$

Note that we obtained the best results by setting $p_{diversity}$=75 and p_{length}=25. It can be seen by this that the referee indeed helps to avoid local optima (see section 3.4.1), because the similarity of tours to each other is taken into account.

The number of individuals which are selected by the referee for the non-winners is set to approximately 10% of the number of generated tours (this depends on the given instance as well as on the generation time).

Table 6 gives the results of the LO2 team. Column 2 gives the length of the best tour of the two concerned experts in the single runs; the creation time can be found in column 3. As for the team, the best generated tour can be found in column 4 along with the creation time in column 5. Column 6 gives the time in which the best tour of the single runs was found by the team. If not exactly this tour was generated, but an even better one, the creation time is given by "before than", indicated by a "<" symbol. Finally, column 7 gives the total time of the team run.

TSP	single run	time	team run	time	comp. time	total time
kroB100	*22 141	2.46	*22 141	5.28	5.28	1:20.22
gr202	*40 160	20.46	*40 160	21.98	21.98	6:59.75
lin318	42 203	2:24.83	42 203	1:39.82	1:39.82	2:22.67
pcb442	50 991	5:20.53	*50 778	22:05.34	< 5:38.41	34:35.15
att532	27 731	21:39.44	27 705	15:42.04	< 11:48.30	44:02.87
ali535	203 011	15:26.52	202 678	15:13.94	< 8:53.18	22:21.45
d657	49 084	31:02.50	49 000	23:34.00	< 19:26.19	39:05.53
gr666	295 731	31:26.12	294 435	25:29.50	< 15:15.27	1h 17:41.13
u724	42 080	1h 47:23.76	41 994	1h 34:17.40	< 28:31.28	1h 49:16.91
rat783	8 816	1h 47:50.97	8 829	1h 19:20.33	-	1h 54:43.88
dsj1000	18 749 543	2h 33:37.87	18 740 224	1h 56:31.66	< 1h 11:11.90	2h 30:04.47
u1060	225 917	1h 01:40.47	224 324	1h 24:11.64	< 38:05.48	1h 25:42.13

Table 6: MUT4_LO2 & ER_LO2: results of team

As can be seen in table 6, for all of the TSP instances (with one exception: rat783), a better tour or at least an equal one is found by the team. For the instance pcb442, the optimal tour could be found for the first time. The majority of team runs (eight runs) show better results than the single ones. Moreover the best tours found by an expert working alone could be found much faster by a team in most of the cases; for example for the instance u724, this tour could be found almost four times faster than by the best expert working alone.

4.2.2 3-Opt Team

In the 3-Opt team we used the following parameters (the parameters for the experts are the same ones as in the best single runs). The referee consists of the functions DIVERSITY and LENGTH; we used the following parameters:

DIVERSITY :

$p_{diversity} = 75$

$p_{length} = 25$

LENGTH :

$num =$ about 10% of the number of generated tours

$p_{shortest} = 0$

$p_{avg\_all} = 0$

$p_{avg\_shortest} = 0$

$p_{avg\_longest} = 0$

$p_{avg\_unsimilar} = 1000$

$p_{gen} = 0$

Similar to the 2-Opt team, the given settings of the parameters $p_{diversity}$ and p_{length} showed the best results. Moreover, for this team, preferring the average length of the most unsimilar tours (compared to the shortest one) with the parameter $p_{unsimilar}$ gave the best results. It can be seen that the referee helps to avoid local optima; TEAMWORK controls the whole optimization process in a way much better than a simple parallel method (with no referees) could do.

The number of selected individuals of the non-winners is set to approximately 10% of the number of created tours in a generation; this depends on the given TSP instance and on the generation time.

The results of the 3-Opt team can be found in Table 7.

Again (with one exception: dsj1000), the team produced better tours or at least tours of the same quality as the experts working alone. For several instances, the best tour of the single runs could be found faster, for example about three times faster for the instance pcb442. But the speed-ups are definitely not as good as in case of the 2-Opt team. The optimal tour could be found for the instance lin318, which none of the experts working alone could achieve. Again, this shows that indeed TEAMWORK helps the experts working together in cooperation with each other.

4.3 Analysis of the Results

The first important observation from our experiments is that for most of the examples (exception: rat783) there is a team that finds a better solution than all the single

TSP	single run	time	team run	time	comp. time	total time
kroB100	*22 141	11.97	*22 141	5.89	5.89	1:05.19
gr202	*40 160	1:41.49	*40 160	1:11.40	1:11.40	10:13.44
lin318	42 071	2:35.06	*42 029	10:08.26	< 4:05.07	11:30.10
pcb442	50 795	1h 29:38.14	50 785	1h 08:20.87	< 29:44.08	1h 36:09.49
att532	27 704	41:40.81	27 703	2h 11:25.30	2h 11:25.30	2h 34:37.75
ali535	202 529	42:21.14	202 506	45:15.39	< 45:15.39	1h 03:56.25
d657	48 975	2h 23:30.94	48 927	2h 23:44.06	< 2h 06:45.13	3h 06:48.15
gr666	294 700	2h 38:57.04	294 522	2h 55:56.31	< 2h 27:07.48	2h 58:17.28
u724	42 001	3h 19:20.56	41 956	3h 17:55.61	< 2h 44:00.87	3h 34:23.16
rat783	8 846	2h 43:47.24	8 836	3h 26:18.32	< 3h 08:50.82	3h 38:02.02
dsj1000	18 751 750	6h 28:45.45	18 768 891	6h 22:30.42	-	6h 35:07.54
u1060	224 909	6h 10:46.95	224 464	5h 58:08.07	< 4h 03:37.92	6h 33:44.48

Table 7: MUT4_LO3 & ER_LO3: results of team

experts (or at least the same one, if it is the optimal solution). Also, again with one exception, the teams find better solutions than the experts that were members of the team. The improvements may be small, but this had to be expected, because the solutions found by the single experts are so near to the optimum.

If we look at the run times for solutions comparable to the best ones found by the single experts then the results of the 2-Opt team are quite impressive. For the hard problems (with the one exception) all speed-ups are greater than 2 with a peak of 4 (using only two processors). The results of the 3-Opt team are not as good. The reasons for this are first that the single experts using 3-Opt as local optimization are not as good as the 2-Opt ones (with respect to run time) and second that the number of new individuals produced in the working periods was significantly smaller for the 3-Opt team. This way, not enough general diversity was provided, so that much smaller regions of the search space were explored. Naturally, this results in less opportunities for cross-fertilization between the experts and so in fewer synergetic effects.

In this context we want to emphasize the importance of the referees. By selecting tours that differ very much from each other (with fitness being only an additional criterion), we are able to have populations that have much more potential than populations that are produced by a single expert alone. Note that this is not easy to integrate into fitness functions or into the general scheme of genetic algorithms. As already stated, using referees that did not put enough interest into diversity, the teams did not behave as good as the ones reported here.

5 Conclusion and Future Work

In this paper, we reported about the application of our TEAMWORK method on solving the Traveling Salesman Problem using genetic algorithms. The resulting distributed system DOTT achieved synergetic speed-ups in generating solutions comparable to those found by single sequential genetic algorithms, even if those algorithms were aug-

mented by local optimization techniques. If one is only interested in the quality of the found solutions, then DOTT was also superior to the single algorithms in most cases.

Our experiments showed that one important reason for the success of TEAMWORK was the referee concept that led to populations with much more diversity than those generated by single genetic algorithms. A selection (by referees) based only on fitness was not as successful as a selection that mainly used diversity. Very likely this will also be the case if one is interested in solving other search problems using genetic algorithms.

Future work will focus on two directions: as already stated, TEAMWORK is a distribution method for search by extension and focus of which genetic algorithms is one instance. Therefore we will apply TEAMWORK to other systems that use genetic algorithms as problem solving method. In addition, TEAMWORK offers very good possibilities to include other search processes than genetic algorithms into such systems. Such an integration of genetic algorithms and other search paradigms is of high interest in many applications. TEAMWORK also allows to use more and other knowledge than in conventional genetic algorithms. As already stated, aspects like diversity can be easily integrated without destroying the necessary balance between randomness, knowledge and success orientation that is required by evolutionary approaches.

With respect to DOTT there remains some work to do. Firstly, the realization of the supervisor is not satisfying. With more experts, specialists and referees, planning capabilities and self-adaptation to the given problem are necessary and have to be provided by the supervisor. Then it will be possible to combine the 2-Opt and 3-Opt teams by starting with experts of the 2-Opt team and switching to the 3-Opt team from time to time later. Also experts that only employ local optimizations can be used towards the end of a team run. More experts will also allow to use more processors in team runs.

References

[Cr58] **Croes, A.**: *A Method for Solving Traveling-Salesman Problems*, Oper.Res. 6, 1958, pp. 791-812.

[De95] **Denzinger, J.**: *Knowledge-Based Distributed Search Using Teamwork*, Proc. ICMAS-95, San Francisco, AAAI-Press, 1995, pp. 81-88.

[DK94] **Denzinger, J. ; Kronenburg, M.**: *Planning for distributed theorem proving: The team work approach*, Proc. KI-96, Dresden, LNAI 1137, 1996, pp. 43-56.

[DL96] **Denzinger, J. ; Lind, J.**: *TWlib - a Library for Distributed Search Applications*, Proc. ICS'96AI, Kaohsiung, 1996, pp. 101–108.

[FM96] **Freisleben, B.; Merz, P.**: *New Genetic Local Search Operators for the Traveling Salesman Problem*, Parallel Problem Solving from Nature - PPSN IV, September 1996, Springer, pp. 890-899.

[Ho92] **John H. Holland**: *Adaption in natural and artificial systems*, Ann Arbor: University of Michigan Press, 2nd edition 1992.

[HK70] **Held, M.; Karp, R.M.**: *The traveling-salesman problem and minimum spanning trees*, Oper.Res. 18, 1970, pp. 1138-1162

[Jo90] **Johnson, D.S.**: *Local Optimization and the Traveling Salesman Problem*, Proc. ALP, LNCS 443, 1990, pp. 446-461.

[Leo95] **Leopold, T.**: *Verteilte Lösung des Travelling Salesman Problems durch Teamwork*, Master Thesis, University of Kaiserslautern, 1995.

[Lin65] **Lin, S.**: *Computer solutions of the traveling salesman problem*, Bell Syst. Tech. J. 44, 1965, pp. 2245-2269.

[LLRS85] **Lawyer, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G.; Shmoys, D.B.**: *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 1985.

[Mi92] **Michalewicz, Z.**: *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Artificial Intelligence 1992.

[Re91] **Reinelt, G.**: *TSPLIB - A Traveling Salesman Problem Library*, ORSA Journal on Computing 3, 1991, pp. 376-384.

[Re94] **Reinelt, G.**: *The Traveling Salesman Problem. Computational Solutions for TSP Applications.*, Springer 1994.

[RSL74] **Rosenkrantz, D.; Stearns, R.; Lewis, P.**: *Approximate Algorithms for the Traveling Salesperson Problem*, Proc. 15th Annual IEEE Symposium of Switching and Automata Theory, 1974, pp 33-42.

[Sch96] **Scholz, Stephan**: *Experten und Gutachter für DOTT*, Project Thesis, University of Kaiserslautern, 1996.