On the complexity and approximability of optimization problems with Minimum Quantity Constraints

Alexander Sieber

Gutachter: **Prof. Dr. Sven O. Krumke**
Prof. Dr. Sebastian Stiller

Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Datum der Verteidigung: 18. August 2020

D 386

Abstract

During the last couple of years, there has been a variety of publications on the topic of *minimum quantity constraints*. In general, a minimum quantity constraint is a lower bound constraint on an entity of an optimization problem that only has to be fulfilled if the entity is "used" in the respective solution. For example, if a minimum quantity q_e is defined on an edge e of a flow network, the edge flow on e may either be 0 or at least q_e units of flow.
Minimum quantity constraints have already been applied to problem classes such as flow, bin packing, assignment, scheduling and matching problems. A result that is common to all these problem classes is that in the majority of cases problems with minimum quantity constraints are NP-hard, even if the problem without minimum quantity constraints but with *fixed* lower bounds can be solved in polynomial time. For instance, the maximum flow problem is known to be solvable in polynomial time, but becomes NP-hard once minimum quantity constraints are added.
In this thesis we consider flow, bin packing, scheduling and matching problems with minimum quantity constraints. For each of these problem classes we provide a summary of the definitions and results that exist to date. In addition, we define new problems by applying minimum quantity constraints to the maximum-weight b-matching problem and to open shop scheduling problems. We contribute results to each of the four problem classes: We show NP-hardness for a variety of problems with minimum quantity constraints that have not been considered so far. If possible, we restrict NP-hard problems to special cases that can be solved in polynomial time. In addition, we consider approximability of the problems: For most problems it turns out that, unless P=NP, there cannot be any polynomial-time approximation algorithm. Hence, we consider *bicriteria* approximation algorithms that allow the constraints of the problem to be violated up to a certain degree. This approach proves to be very helpful and we provide a polynomial-time bicriteria approximation algorithm for at least one problem of each of the four problem classes we consider. For problems defined on graphs, the class of series parallel graphs supports this approach very well.
We end the thesis with a summary of the results and several suggestions for future research on minimum quantity constraints.

Im Laufe der letzten Jahre sind zahlreiche Veröffentlichungen erschienen, die sich mit dem Thema *Minimum-Quantity-Bedingungen* befassen. Allgemein beschreibt eine Minimum-Quantity-Bedingung eine untere Schranke, die sich auf ein Objekt eines Optimierungsproblems bezieht. Diese untere Schranke muss nur dann erfüllt sein, wenn eine Lösung das jeweilige Objekt "nutzt". Wenn zum Beispiel eine Minimum-Quantity q_e für eine Kante e eines Flussnetzwerks definiert ist, darf der Fluss auf dieser Kante entweder 0 oder mindestens q_e Flusseinheiten betragen.
Minimum-Quantity-Bedingungen wurden bereits auf Problemklassen wie Fluss-, Behälter-, Assignment-, Scheduling- und Matchingprobleme angewendet. Für all diese Problemklassen gilt, dass die meisten Probleme mit Minimum-Quantity-Bedingungen NP-schwer sind, selbst wenn die Probleme ohne Minimum-Quantity-Bedingungen aber mit *festen* unteren Schranken in polynomieller Zeit lösbar sind. Das Maximaler-Fluss-Problem ist bekanntermaßen in polynomieller Zeit lösbar, wird aber durch das Hinzufügen von Minimum-Quantity-Bedingungen NP-schwer.
In dieser Arbeit betrachten wir Fluss-, Behälter-, Scheduling- und Matchingprobleme mit Minimum-Quantity-Bedingungen. Für jede dieser Problemklassen fassen wir die bisherigen Definitionen und Ergebnisse zusammen. Außerdem wenden wir Minimum-Quantity-Bedingungen auf das Maximum-Weight b-Matching-Problem und auf Open-Shop-Scheduling-Probleme an. Wir tragen zu jeder der vier Problemklassen neue Ergebnisse bei: Wir zeigen für mehrere bisher nicht betrachtete Probleme mit Minimum-Quantity-Bedingungen, dass diese NP-schwer sind. Wenn möglich, schränken wir NP-schwere Probleme weiter ein, um Spezialfälle zu erhalten, die in polynomieller Zeit gelöst werden können. Außerdem betrachten wir die Approximierbarkeit der Probleme: Für die meisten Probleme stellt sich heraus, dass diese, sofern nicht P=NP gilt, nicht in polynomieller Zeit approximiert werden können. Daher untersuchen wir auch die *bikriterielle* Approximierbarkeit, bei der Nebenbedingungen zu einem gewissen Grad verletzt sein dürfen. Dieser Ansatz erweist sich als sehr nützlich und wir sind in der Lage für mindestens ein Problem jeder Problemklasse einen bikriteriellen Approximationsalgorithmus mit polynomieller Laufzeit anzugeben. Für Probleme, die auf Graphen definiert sind, unterstützen seriell-parallele Graphen diesen Ansatz besonders gut.
Abschließend fassen wir die Ergebnisse dieser Arbeit zusammen und schlagen einige Ansätze vor, die den Ausgangspunkt für die zukünftige Forschung in diesem Gebiet bilden könnten.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Objectives

During the last couple of years, there has been a variety of publications on the topic of *minimum quantity constraints*. In general, a minimum quantity constraint is a lower bound constraint on an entity of an optimization problem that only has to be fulfilled if the entity is "used" in the respective solution. If the entity is not used, the lower bound constraint can be ignored. For example, minimum quantity constraints can be applied to edges of flow networks: If a minimum quantity q_e is defined on the edge e, the edge flow on e may either be 0 or at least q_e units of flow.
Minimum quantities have been applied to a variety of problem classes such as flow [14, 23, 25, 31, 32, 44, 45, 47, 48], bin packing [45], assignment [33], scheduling [21, 45] and matching problems [2, 3, 10, 26, 39]. A result that is common to all these classes is that most problems that are solvable in polynomial time become NP-hard after adding minimum quantities.
The objective of this thesis is to continue the research on the topic of minimum quantities and to provide new algorithmic and complexity results. We provide an overview of previous work and results related to minimum quantities and contribute to the topic by improving several of these results and by defining and analyzing new problems with minimum quantities. Our analysis usually starts with classifying problems regarding the complexity classes P and NP. Depending on this classification we search for exact algorithms that run in polynomial time, exact pseudo-polynomial-time algorithms or polynomial-time approximation algorithms or approximation schemes. The implementation and benchmarking of algorithms is not within the scope of this thesis.

1.2 Structure of the thesis

The thesis has been structured as modular as possible by reducing the interdependencies between the chapters to a minimum. Chapter 1 introduces the topic of minimum quantities and provides basic definitions and results from complexity theory and from graph theory that are used in the subsequent chapters. The introductory sections on complexity theory and on graph theory are partly similar to the corresponding contents of [45].
Each of the following chapters focuses on a specific class of optimization problems. These chapters can be read independently from each other and are structured as follows:

- Chapter 2: Bin Packing problems
- Chapter 3: Scheduling problems
- Chapter 4: Matching problems
- Chapter 5: Flow problems

Each of the chapters 2, 3, 4 and 5 has its own introduction and conclusion. This structure supports the modularity of the thesis. The introduction to each of the problem-specific chapters provides an overview of previous work on the respective problem. Finally, chapter 6 provides an overall conclusion.

1.3 Complexity theory fundamentals

This section summarizes basic definitions and results from complexity theory as well as notation conventions that are used throughout this thesis.
We assume that the reader is familiar with complexity theory and combinatorial optimization in general. Otherwise, we refer the reader to [16] for a comprehensive introduction to the classes P and NP and to NP-completeness, including a list of NP-complete problems. In addition, an introduction to the class of *strongly* NP-complete problems can be found in [15]. Moreover, [12] provides an overview of combinatorial optimization in general. For the sake of convenience, we do not discuss complexity theory in terms of languages and Turing machines, but use definitions that refer to *problems* instead.
As the notion of strong NP-hardness is crucial throughout the thesis, we summarize the corresponding definitions and results here. The authors of [16] define two functions that are associated with every decision problem: Length[I] describes the length of an encoded problem instance I and Max[I] corresponds to its largest numerical value, such as the capacity of a bin in the bin packing problem or the capacity of an edge in a flow problem. The results

given in [16] remain valid for all functions Length[I] and Max[I] that are "polynomially related" to the ones used in [16]. We refer the reader to [16] for further information on the formal definitions of the functions Length[I] and Max[I].
In [16] the following definition is given:

Definition 1.3.1 (number problem [16])**.** A decision problem Π is a number problem if there exists no polynomial p so that $\text{Max}[I] \leq p(\text{Length}[I])$ for all instances I of Π.

Note that most problems in this thesis are in fact number problems.
As usual, in this thesis problem instances are assumed to be encoded in binary. In particular, the space required for their representation is logarithmical in the size of the numerical values of the problem instance. Unless specified otherwise, "log" denotes the logarithm function with base 2 throughout this thesis.
Before we proceed with defining strong NP-completeness, we introduce the following notation [16]:

Notation 1.3.2. Given a polynomial p (over the integers), Π_p denotes the restriction of the decision problem Π to instances for which $\text{Max}[I] \leq p(\text{Length}[I])$.

With the above definitions we are now ready to define strong NP-completeness:

Definition 1.3.3 (strong NP-completeness [16])**.** A decision problem Π is strongly NP-complete if it belongs to NP and there exists a polynomial p (over the integers) such that Π_p is NP-complete.

Instead of applying Definition 1.3, we use reductions from already known strongly NP-complete problems in order to show strong NP-completeness. These reductions are called pseudo-polynomial transformations and are formally defined in [16].
The previous definitions only deal with decision problems. The notion of search problems, as given in [16], generalizes decision and optimization problems. Note that the above definition of NP-completeness requires membership in NP, which consists only of decision problems. So, as usual, we refer to search problems as NP-hard if there is an NP-complete (or, by transitivity, an NP-hard) problem that Turing-reduces to the given problem [16]. The notion of strong NP-completeness can be generalized to strong NP-hardness in an obvious way. Given that we focus on optimization problems, we state the following results in terms of NP-*hardness*.

Definition 1.3.4 (pseudo-polynomial-time algorithm [15])**.** A pseudo-polynomial-time algorithm is an algorithm that runs in time bounded by a polynomial in the two variables Length[I] and Max[I].

The following theorem shows how strong NP-hardness and the existence of pseudo-polynomial algorithms are related:

Theorem 1.3.5 ([15])**.** *If a problem is strongly NP-hard, it cannot be solved by a pseudo-polynomial algorithm, unless P=NP.*

If, conversely, there is a pseudo-polynomial algorithm for a problem, we usually refer to the problem as *weakly* NP-hard. If it is not clear whether a problem is weakly or strongly NP-hard or if this distinction is not relevant in a certain context, we simply refer to the problem as NP-hard. The term *weak NP-completeness* is used analogously.
There is also a connection between the approximability of a problem and its complexity, as Theorem 1.3.8 shows. We first need the following definitions:

Definition 1.3.6 (polynomial-time α-approximation algorithm [33])**.** Given a maximization problem, a polynomial-time α-approximation algorithm requires a number of steps that is polynomially bounded in the encoding size of the given instance of the problem to achieve the following: If there is a feasible solution to the given instance, the algorithm computes a feasible solution with objective value at least $\frac{1}{\alpha}$ times the optimal objective value. Otherwise, the algorithm outputs infeasibility of the given instance. For a minimization problem the definition of a polynomial-time α-approximation algorithm works analogously.

Note that, according to this definition, a polynomial-time α-approximation algorithm can be used to determine if there is a feasible solution to the given problem instance.

Definition 1.3.7 (fully polynomial-time approximation scheme [16])**.** For a given problem, a family of algorithms consisting of a polynomial-time $(1+\epsilon)$-approximation algorithm A_ϵ for every fixed $\epsilon > 0$ is called a polynomial-time approximation scheme (PTAS). If, in addition, the running time of every A_ϵ is polynomially bounded in the encoding size of the input instance and in $\frac{1}{\epsilon}$, the family of algorithms is called a fully polynomial-time approximation scheme (FPTAS).

Theorem 1.3.8 ([16])**.** *Let a strongly NP-hard problem be given. If there is a two-variable polynomial q so that for all instances I of the problem the optimal objective value is bounded by q(Length[I], Max[I]), then, unless P=NP, there cannot exist any FPTAS for the problem.*

For the problems that we consider in the following chapters the objective functions are bounded by such a polynomial in Length[I] and Max[I]. Hence, if the problems are strongly NP-hard, Theorem 1.3.8 can be applied.

We obtain an alternative approach to approximation by allowing a certain relaxation of the constraints:

Definition 1.3.9 (polynomial-time (α, β)-approximation algorithm [33])**.** By a polynomial-time (α, β)-approximation algorithm for a maximization problem we mean an algorithm that, for any given instance of the problem, achieves the following in a number of steps that is polynomially bounded in the encoding length of the respective instance: If the instance admits a feasible solution, the algorithm computes a solution that violates a certain given subset of the constraints by at most a factor $\beta \geq 1$ and the objective value of which is at least $\frac{1}{\alpha}$ times as large as the objective value of an optimal solution that satisfies the constraints strictly. If the instance does not admit a feasible solution, the algorithm outputs infeasibility of the instance. An analogous definition can be applied to the case of minimization problems.

In some cases we do not specify α and β and simply refer to (α, β)-approximation algorithms as *bicriteria* approximation algorithms.
We have already pointed that an α-approximation algorithm according to Definition 1.3.6 can be used to find out if there is a feasible solution to a given problem instance. Note that the same holds for polynomial-time (α, β)-approximation algorithms according to Definition 1.3.9.
Some authors allow bicriteria (or multicriteria) approximation algorithms to generate approximate solutions even if the underlying instance does not admit any feasible solution [28, 35, 36, 42]. A similar approach is used in [24]: *Dual* approximation algorithms allow a certain degree of infeasibility of the solution while its objective value is required to be (super-)optimal.

Remark 1.3.10. Our analysis regarding the existence of bicriteria approximation algorithms for several problems in the following chapters is based on Definition 1.3.9. Using one of definitions used in the previously mentioned publications might lead to different results.

Unless specified otherwise, β in Definition 1.3.9 refers to the minimum quantity constraints and, if applicable, capacity constraints of the respective problem. In the case of flow problems, for example, it refers to the minimum quantity and capacity constraints defined on the edges, while it refers to the minimum quantity constraints defined on the load of the machines in scheduling problems. Note that we do *not* allow (α, β)-approximation algorithms for flow problems to generate solutions that violate the flow conservation constraints.

Throughout this thesis we assume $\mathbb{N}$ to consist of the *non-negative* integers. We denote the set of *strictly positive* integers by $\mathbb{N}_+$. Analogously, we denote the sets of *strictly positive* rational and real numbers by $\mathbb{Q}_+$ and $\mathbb{R}_+$, respectively.

1.4 Graph theory fundamentals

In this chapter we provide a brief overview of the notation we use regarding problems on graphs and of two classes of graphs that prove to be helpful for solving problems with minimum quantity constraints.
A graph G is given by a set of nodes V and a set of directed or undirected edges E. We use the notation $G = (V, E)$. If our analysis involves multiple graphs, we sometimes use the notation $V(G)$ and $E(G)$ in order to point out that these sets of nodes and edges belong to the graph G. If there is exactly one undirected edge $e \in E$ connecting the nodes $v_1, v_2 \in V$, we sometimes denote it by (v_1, v_2) or equivalently by (v_2, v_1). Analogously, if a directed edge $e \in E$ points from node v_1 to node v_2, we sometimes refer to it as (v_1, v_2). We denote the set of edges that are incident to exactly one node in a set of nodes $V' \subseteq V$ by $\delta(V') \subseteq E$. In a directed graph, we denote the subset of $\delta(V')$ consisting of those edges *pointing to* one of the nodes in V' by $\delta^-(V')$ and the subset of edges *starting at* one of the nodes in V' by $\delta^+(V')$. For a single node v we denote these sets by $\delta(v)$, $\delta^-(v)$ or $\delta^+(v)$, respectively. Sometimes, in order to highlight the graph that we are referring to, we add the name of the graph, e.g. $\delta^+_G(V')$.
Given a graph $G = (V, E)$ and a subset of the nodes $V' \subseteq V$, we denote the subgraph of G that consists of the nodes V' and of the edges $\{e = (v_1, v_2) \in E \,|\, v_1, v_2 \in V'\}$ by $G[V']$. We call $G[V']$ the subgraph of G induced by V'.
We now provide the definitions of two classes of graphs: Series parallel graphs and graphs with bounded treewidth. Series parallel graphs are defined as follows:

Definition 1.4.1 (series parallel graph [6])**.** A series parallel graph is a triple (G, s, t) with $G = (V, E)$ a graph and $s, t \in V$ such that one of the following cases holds:

- G has two nodes, s and t, and one edge between s and t.
- There are series parallel graphs $(G_1, s_1, t_1), \ldots, (G_r, s_r, t_r)$, such that (G, s, t) can be obtained by a parallel composition of these graphs: G is obtained by taking the disjoint union of $G_1, \ldots, G_r$, identifying all nodes $s_1, \ldots, s_r$ to s, and identifying all nodes $t_1, \ldots, t_r$ to t.
- There are series parallel graphs $(G_1, s_1, t_1), \ldots, (G_r, s_r, t_r)$, such that (G, s, t) can be obtained by a series composition of these graphs: G is obtained by taking the disjoint union of $G_1, \ldots, G_r$, identifying for $i = 1, \ldots, r-1$, node t_i with node s_{i+1} and letting $s := s_1$ and $t := t_r$.

Sometimes these graphs are also referred to as *two-terminal* series parallel graphs.
Note that there might me more than one sequence of compositions to construct a given series

parallel graph. A sequence of compositions can be represented by a so-called binary sp-tree. Every leaf node corresponds to a single-edge graph and the root node corresponds to the graph to be constructed. The nodes in between correspond to series parallel graphs that are obtained by series and parallel compositions. A binary sp-tree can be computed efficiently [6]:

Lemma 1.4.2. *Given a series parallel graph, a corresponding binary sp-tree can be computed in polynomial time.*

Note that the number of series parallel graphs that occur during the sequence of compositions is bounded by $O(|E(G)|)$ if G denotes the series parallel graph to be constructed.
In the context of *directed* graphs (in particular, in the context of flow networks), the edges of series parallel graphs are directed from the source to the sink. Note that this implies that the graph is acyclic.

Treewidth-bounded graphs are the second class of graphs that we briefly introduce in this section. A comprehensive introduction to treewidth and related results can be found in [4] and in [29]. Graphs for which the treewidth is bounded by a constant are particularly interesting because problems on such graphs can often be solved in polynomial time using a dynamic programming approach, even if the problem is NP-hard in general [7].

Definition 1.4.3 (tree-decomposition [4])**.** A tree-decomposition of a graph $G = (V, E)$ is a pair $D = (\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a family of subsets of V, one for each node of T, and T a tree such that

a) $\bigcup_{i \in I} X_i = V$.

b) For all edges $(v, w) \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.

c) For every node $v \in V$ the subset of nodes $\{i \in I | v \in X_i\}$ of T induces a subtree of T.

Alternatively, we can use one of the following equivalent properties instead of property c):

c') If $v \in X_i$ and $v \in X_k$ for some $v \in V$, then we have $v \in X_j$ for all $j \in I$ for which node j is on the unique path from i to k in T.

c") For all $i, j, k \in I$: If j is on the unique path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

Based on this structure we can define two properties, one of which is related to a given tree-decomposition and the other one to the underlying graph:

Definition 1.4.4 (width of a tree-decomposition [4])**.** The width of a tree-decomposition D as defined above is $\text{width}(D) := \max_{i \in I} |X_i| - 1$.

Definition 1.4.5 (treewidth [4])**.** Given a graph G, its treewidth is $\text{tw}(G) := \min\{\text{width}(D) \mid \text{D is a tree-decomposition of } G\}$.

For graphs with bounded treewidth a tree-decomposition can be computed in linear time as the following theorem shows [5, 29]:

Theorem 1.4.6. *Given a graph G, there is an algorithm that computes a tree-decomposition of G that has width $t = tw(G)$ in $2^{O(t^3)} \cdot |V(G)|$ steps.*

In most cases we will use "nice" tree-decompositions that have a special structure:

Definition 1.4.7 (nice tree-decomposition [29])**.** A nice tree-decomposition is a tree-decomposition where a root of the corresponding tree T can be chosen in such a way that T is a rooted binary tree and each node i of T corresponds to one of the following types:

- Leaf node: If i does not have any child nodes, then it is called a leaf node.
- Insert node: If i has exactly one child node j and $X_i = X_j \cup \{v\}$ for some $v \notin X_j$, then it is called an insert node.
- Forget node: If i has exactly one child node j and $X_i = X_j \setminus \{v\}$ for some $v \in X_j$, then it is called a forget node.
- Join node: If i has exactly two child nodes j_1 and j_2 and $X_i = X_{j_1} = X_{j_2}$, then it is called a join node.

In the following we use the notation given in [29]: We denote the union of X_i and of all X_j, where j is a descendant of i, by V_i. For each i the set V_i induces a subgraph $G_i := G[V_i]$. We set $E_i := E(G_i)$.
A nice tree-decomposition can be computed from a given tree-decomposition in polynomial time, as the following theorem shows [29]:

Theorem 1.4.8. *Given a tree-decomposition D for some graph G with treewidth t where T denotes the corresponding tree, we can construct a nice tree-decomposition D' for G that fulfills width(D') $\leq$ width(D) and $|V(T')| \leq \text{width}(D) \cdot |V(T)|$ and where T' denotes the binary tree of D' in polynomial time.*

The following result from [43] shows that we can even determine a nice tree-decomposition in *linear* time if the treewidth is bounded by a constant t:

Theorem 1.4.9. *There is an algorithm that constructs a nice tree-decomposition of width t and size $O(|V(G)|)$ for a graph G with treewidth at most t in linear time.*

We provide one more result from [29]:

Lemma 1.4.10. *Let a graph $G = (V, E)$ and a nice tree-decomposition be given, where T denotes the binary tree.*

- *Let $i \in V(T)$ an insert node and j its child node in T so that $X_i = X_j \cup \{v\}$. The graph G_i can be constructed from G_j by adding the node v and all edges that are incident to it in G. In particular, v is not adjacent to any node in $V_j \setminus X_j$.*
- *Let $i \in V(T)$ a forget node and j its child node in T so that $X_i = X_j \cup \{v\}$. Then G_i and G_j are identical.*
- *Let $i \in V(T)$ a join node and j_1 and j_2 its child nodes in T so that $X_i = X_{j_1} = X_{j_2}$. For $v \in V_{j_1} \setminus X_i$, $w \in V_{j_2} \setminus X_i$ we have that $(v, w) \notin E$.*

Series parallel graphs are closely connected to the notion of treewidth as the following lemma shows [8]:

Lemma 1.4.11. *Series parallel graphs have treewidth at most 2.*

We end this section by providing the definition of *complete* binary trees:

Definition 1.4.12 (complete binary tree [13])**.** Let h be a positive integer. A complete binary tree of height h is a binary tree for which the following holds:

- All leaf nodes have distance h to the root node.
- All non-leaf nodes have exactly two child nodes.

Chapter 2

Bin packing problems

2.1 Introduction

The bin packing problem is a well-known problem in combinatorial optimization. An instance of the decision version of the problem is given by a set of items of different sizes, a bin capacity b and a number k. The problem is to decide whether all items fit into at most k bins without exceeding the capacity b for any of the bins. This problem is known to be strongly NP-complete [16].
Generalized optimization versions of this problem are defined in [33] and in [45], where minimum quantity constraints are applied to the bins. In [33] the so-called generalized assignment problem with minimum quantities (GAP-MQ) is defined. In [45] a special case of GAP-MQ, the generalized bin packing with minimum quantities (GBPMQ), is defined. The formal definitions of GAP-MQ and GBPMQ as well as a summary of the results from [33] and [45] are given in the next section.

2.2 Basics and Definitions

The following definition of the generalized assignment problem with minimum quantities is given in [33]:

Definition 2.2.1 (generalized assignment problem with minimum quantities (GAP-MQ))**.** An instance of GAP-MQ is given by n items, m bins, bin capacities $b_1, \ldots, b_m \in \mathbb{N}$ and minimum quantities $q_1, \ldots, q_m \in \mathbb{N}$ with $q_j \leq b_j$ for all $j = 1, \ldots, m$. An item size $s_{i,j} \in \mathbb{N}$ and a profit $p_{i,j} \in \mathbb{N}$ are associated with assigning item i to bin j for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$. The task is to find an assignment of a subset of the items to a subset of the bins that maximizes the sum of the profits and for which the space used in each bin j is 0 or between q_j and b_j.

Additionally, the following special cases of GAP-MQ are defined in [33]:

Definition 2.2.2 (seminar assignment problem (SAP))**.** The seminar assignment problem consists of all instances of GAP-MQ, where $s_{i,j} = 1$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

Definition 2.2.3 (participant maximization problem (PMP))**.** The participant maximization problem consists of all instances of GAP-MQ, where $p_{i,j} = s_{i,j} = 1$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

In [33], GAP-MQ is shown to be strongly NP-hard and non-approximable in polynomial time (unless P=NP), even in the case of unit profits or if $p_{i,j} = s_{i,j}$ for all i, j. This still holds if the item sizes do not depend on the bins the items are packed into. The non-approximability result even holds if we restrict the problem definition to only one bin. On the other hand, polynomial solvability is shown for the special case where profits do not depend on the bins and the maximum bin capacity and the number of different item types are fixed. If the number of bins is fixed and there is some $\delta > 1$ so that $q_j \geq \delta s_{i,j}$ for all i, j, then there is a polynomial-time algorithm that computes a solution for which the minimum quantities are violated by at most a factor $1 - 1/\delta$, the capacities are violated by at most a factor $1 + 1/\delta$ and the value of the solution is at least as large as the value of an optimal feasible solution to the problem. For $\delta = 2$, computational results of an implementation of the algorithm are provided. According to these results, the minimum quantities and capacities are, on average, violated only by a rather small factor compared to the theoretical bounds. If the number of bins is fixed, there is a pseudo-polynomial dynamic program for GAP-MQ.
SAP is strongly NP-hard and there is no PTAS (unless P=NP), even if $p_{i,j} \in \{0, 1\}$ for all i, j. However, if the number of bins is fixed, it can be solved in polynomial time.
If only unit sizes and unit profits are allowed, the problem becomes easier: PMP is *weakly* NP-hard and a FPTAS as well as a greedy 2-approximation algorithm are shown.

In [45], another variant of GAP-MQ is considered:

Definition 2.2.4 (generalized bin packing problem with minimum quantities (GBPMQ))**.** The generalized bin packing problem with minimum quantities consists of all instances of GAP-MQ, where $s_{i,j} = s_i$ for some $s_i \in \mathbb{N}$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

Note that SAP and PMP are special cases of GBPMQ so that all the above hardness results regarding SAP and PMP also apply to GBPMQ. On the other hand, GBPMQ is a special case of GAP-MQ so that all (positive) solvability and approximability results regarding GAP-MQ also apply to GBPMQ. Note that according to [33] GAP-MQ cannot be approximated in polynomial time unless P=NP, even for unit profits and if there is only one bin. Such an instance of GAP-MQ can be interpreted as instances of GBPMQ with unit profits and one

bin. Hence, unless P=NP, GBPMQ with unit profits and one bin is also inapproximable in polynomial-time. Trivially, the minimum quantities and capacities are uniform in this case. We conclude that GBPMQ with unit profits, uniform minimum quantities and uniform bin capacities is also inapproximable in polynomial time, unless P=NP.
In [45] SAP (and thus GBPMQ) is shown to remain strongly NP-hard even if the minimum quantities and capacities are restricted to certain special cases. Additionally, the so-called *restricted* generalized bin packing problem with minimum quantities (RGBPMQ) and the *restricted* seminar assignment problem (RSAP) are defined: In these variants of GBPMQ and SAP each item may be assigned only to a subset of the bins. This restriction is characterized by so-called feasibility graphs. The feasibility graph is a bipartite graph where the nodes are partitioned into the sets I (corresponding to items) and B (corresponding to bins). If an item i may be packed into bin j, then there is an edge connecting node i and node j in the feasibility graph. Note that if the feasibility graph is the complete bipartite graph, the problem corresponds to GBPMQ or SAP, respectively. Thus, RGBPMQ and RSAP are generalizations of GBPMQ and SAP. In [45] it is shown that if the feasibility graph is a tree, the problem remains NP-hard and inapproximable, unless P=NP. Approximating RSAP remains strongly NP-hard, even if the maximum number of items that are feasible for each bin is bounded by 3 and the profits $p_{i,j}$ are restricted to values in $\{0,1\}$ for all i, j. RGBPMQ can be solved in polynomial time if it is restricted to those instances where the feasibility graph is a tree, the degree of each of its nodes is bounded by a fixed value k and all $p_{i,j}$ are bounded by a fixed polynomial in the number of edges of the feasibility graph. In the following we assume RPMP to be defined analogously, i.e. it is a variant of PMP where the feasible assignments of items to bins are given by a feasibility graph.

2.3 Complexity Results & Algorithms

We begin this section by proving polynomial-time solvability for a special case of GBPMQ using a dynamic program. Afterwards we use this dynamic program in order to show the existence of a polynomial-time $(1,1+\epsilon)$-approximation algorithm for GBPMQ with uniform minimum quantities, uniform capacities and with profits that do not depend on the bin. As mentioned in the introduction, unless P=NP, there is no polynomial-time approximation for this problem that guarantees feasibility of the solution.

Theorem 2.3.1. *Given a fixed integer $k > 0$, GBPMQ with uniform minimum quantities and uniform bin capacities can be solved in polynomial time if the profits do not depend on the bins and there are at most k different item sizes.*

Proof. We prove the theorem by providing a dynamic program that runs in polynomial time. Assume that an instance of GBPMQ as specified in the theorem is given by n items with profit p_i and item size s_i for each item i. Additionally, there are m bins with uniform minimum quantities q and uniform bin capacities b. W.l.o.g. we assume $n \geq m$.
We classify the items by their respective sizes. By assumption, there are at most k such classes. W.l.o.g. we assume that there are exactly k classes, otherwise empty dummy classes can be created. Let $0 \leq n_c \leq n$ denote the number of items with size s_c for all $c \in \{1, \dots, k\}$.
We denote the powerset of the set of items by A, where each set of items in A is represented by a k-tuple:

$$A \coloneqq \left\{ a \in \mathbb{Z}^k \middle| 0 \leq a_c \leq n_c \text{ for all } c \in \{1, \dots, k\} \right\}$$

We define the following set where each element corresponds to a feasible packing of *one* bin:

$$A_1 \coloneqq \left\{ a \in A \middle| \sum_{c=1}^{k} a_c s_c \in \{0\} \cup [q, b] \right\}$$

The dynamic program iteratively computes the value $f(a, j) \in \{0, 1\}$ for all $a \in A$ and $1 \leq j \leq m$. If $f(a, j) = 1$, this indicates that there is a feasible solution on j bins that uses exactly the items corresponding to a. If $f(a, j) = 0$, then there is no such solution. For all $a \in A$, we initialize the dynamic program for $j = 1$ by setting

$$f(a, 1) \coloneqq \begin{cases} 1 & a \in A_1 \\ 0 & \text{else} \end{cases}$$

We iteratively increase j (until we reach $j = m$) and compute $f(a, j)$ for all $a \in A$ in each iteration as follows:

$$f(a, j) \coloneqq \max_{a' \in A_1} f(a - a', j - 1)$$

The correctness of the computation (i.e. $f(a, j) = 1$ if and only if there is a feasible solution on j bins that uses exactly the items corresponding to a) follows by induction: The base case is $j = 1$ and the correctness follows by construction.
For the induction step, we assume that $f(a, j-1)$ has been computed correctly for all $a \in A$. In order for a feasible packing that consists of the items corresponding to a to exist on j bins, there must be a feasible packing corresponding to some a' that fits into one bin and a feasible packing of the items given by $a - a'$ into exactly $j - 1$ bins (i.e. $f(a - a', j - 1) = 1$). This is exactly how the dynamic program determines $f(a, j)$.
Note that if $f(a, j-1) = 1$, then $f(a, j) = 1$ since the empty packing is contained in A_1. Thus, it is sufficient to consider the combinations of items for which there is a feasible solution on

m bins in order to determine an optimal solution. The value we obtain from packing a set of items corresponding to some $a = (a_1, \ldots, a_k)$ is computed by selecting the a_c items with the highest profits from each class c and summing up the profits. By comparing the values of all combinations of items corresponding to some $a \in A$ for which a feasible packing exists on m bins, an optimal solution can be found. The algorithm computes an optimal solution in polynomial time:

- Grouping the items by size and sorting them by decreasing profit within each group can be done in $O(n \log n)$ steps.
- For computing A_1, it is necessary to check for all possible combinations of items whether packing the respective items into one bin is feasible regarding the minimum quantity and the bin capacity. There are $\prod_{c=1}^{k} (n_c + 1)$ combinations of items, so A_1 can be computed in $O(n^k)$ and A_1 contains at most $\prod_{c=1}^{k} (n_c + 1)$ elements.
- The computational complexity of determining one specific value $f(a, j)$ (if all values required for the computation have already been computed) is $O(n^k)$. There are $m \prod_{c=1}^{k} (n_c + 1)$ possible combinations of a and j. So for computing all $f(a, j)$ we require at most $O(n^{2k+1})$ steps (considering that $m \leq n$).
- Determining the value of a solution given by a vector a as described above can be done in $O(n)$ if the items have been sorted. There are at most $\prod_{c=1}^{k} (n_c + 1)$ combinations of items for which a feasible packing exists. So finding an optimal solution requires at most $O(n^{k+1})$ steps.

All in all, we get that the computational complexity of the algorithm is $O(n^{2k+1})$. Given that k is assumed to be fixed, this is polynomial in the size of the problem. □

We now present a bicriteria (α, β)-approximation algorithm for GBPMQ. In this case, the factor β refers to the minimum quantity constraints and to the bin capacity constraints. The idea of the algorithm consists of rounding item sizes and clustering small items in order to obtain a limited number of different item sizes. Then the above dynamic program is applied to this simplified instance of the problem.

Theorem 2.3.2. *For a fixed precision parameter $\epsilon > 0$ and an instance of GBPMQ with uniform minimum quantities, uniform bin capacities and with profits that do not depend on the bin, there is a polynomial-time $(1, 1 + \epsilon)$-approximation algorithm.*

Proof. We prove the claim by providing the algorithm. As usual, let n and m denote the number of items and bins, respectively, and let the item profits and sizes be given by p_i and s_i for $i \in \{1,\dots,n\}$. The values q and b denote the minimum quantity and the capacity of the bins. Note that we are especially interested in small values of ϵ. Hence, w.l.o.g. we assume $\epsilon \leq 1$ in the following. If $\epsilon > 1$ at the beginning of the algorithm, we update ϵ by setting $\epsilon := 1$. We set $\sigma := \sqrt[3]{(1+\epsilon)} - 1$ so that $(1+\sigma)^3 = 1+\epsilon$. Note that this implies $\sigma < 1$.
The algorithm distinguishes the two cases $q \leq \frac{\sigma}{2}b$ and $q > \frac{\sigma}{2}b$ separately. In each of the cases a threshold for distinguishing "large" and "small" items is defined and the small items are clustered. Afterwards the sizes of the large items and the cluster items are rounded. These new items as well as a new minimum quantity q' and a new capacity b' are used as an input for the dynamic program from 2.3.1.

- Case 1: $q \leq \frac{\sigma}{2}b$
  We partition the items into large and small items, where L denotes the set of large items and S denotes the set of small items:
  $L := \left\{ i \in \{1,\dots,n\} \middle| s_i \geq \frac{\sigma}{2}b \right\}$
  $S := \left\{ i \in \{1,\dots,n\} \middle| s_i < \frac{\sigma}{2}b \right\}$
  If there are any small items we partition them into sets $C_r \subseteq S$ for $r \in \{1,\dots,g\}$ for some integer g in such a way that for each of these sets (possibly except for one) the sum of the respective item sizes is in $\left[\frac{\sigma}{2}b, \sigma b\right)$. The items are partitioned by non-increasing order of the profit densities p_i/s_i, i.e. if $i_1 \in C_{r_1}$ and $i_2 \in C_{r_2}$ and $r_1 < r_2$ then $p_{i_1}/s_{i_1} \geq p_{i_2}/s_{i_2}$.
  For each of the sets C_r we create a corresponding cluster item c_r.
  We set $s_{c_r} := \sum_{i \in C_r} s_i \in \left[\frac{\sigma}{2}b, \sigma b\right)$ and $p_{c_r} := \sum_{i \in C_r} p_i$. Note that if $r_1 < r_2$ then $p_{c_{r_1}}/s_{c_{r_1}} \geq p_{c_{r_2}}/s_{c_{r_2}}$ by construction. By $C := \{c_1,\dots,c_g\}$ we denote the set of cluster items.
  Set $k := \left\lceil \log_{1+\sigma} \frac{2}{\sigma} \right\rceil$. This implies $(1+\sigma)^k \frac{\sigma}{2} \geq 1$. In addition, since $\sigma < 1$, we have $k \geq 2$.
  Additionally we define the following intervals and values:

  - $I_h := \left[(1+\sigma)^{h-1}\frac{\sigma}{2}b, (1+\sigma)^h \frac{\sigma}{2}b\right) \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{h-1}\frac{\sigma}{2}b \right\rceil, \left\lceil (1+\sigma)^h \frac{\sigma}{2}b \right\rceil - 1\right]$ for all $1 \leq h \leq k-1$
  - $I_k := \left[(1+\sigma)^{k-1}\frac{\sigma}{2}b, (1+\sigma)^k \frac{\sigma}{2}b\right] \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{k-1}\frac{\sigma}{2}b \right\rceil, \left\lfloor (1+\sigma)^k \frac{\sigma}{2}b \right\rfloor\right]$
  - $t_h := \left\lceil (1+\sigma)^{h-1}\frac{\sigma}{2}b \right\rceil$ for all $1 \leq h \leq k$

  We now round the sizes of the items in L and C as follows:
  If $s_i \in I_h$ for some $i \in L \cup C$ and $1 \leq h \leq k$, set $s'_i := t_h$. By construction, it is possible that $s_{c_g} < \frac{\sigma}{2}b$. In this case, set $s'_{c_g} := s_{c_g}$.

Set $q' := \left\lceil \frac{q}{(1+\sigma)^2} \right\rceil$ and $b' := \left\lfloor (1+\sigma)^2 b \right\rfloor$. Note that the items sizes s'_i and s'_{c_g} are integral. Hence, rounding q' and b' to integral values does not affect the set of feasible solutions.

- Case 2: $q > \frac{\sigma}{2} b$
  As in case 1, all items are partitioned into small and large items. However, the definitions of the sets S and L are different:
  $L := \left\{ i \in \{1,\dots,n\} \middle| s_i \geq \frac{\sigma^2}{4(1+\sigma)^2} b \right\}$
  $S := \left\{ i \in \{1,\dots,n\} \middle| s_i < \frac{\sigma^2}{4(1+\sigma)^2} b \right\}$
  This time, small items are clustered so that the sizes of the cluster items are in $\left[\frac{\sigma^2}{4(1+\sigma)^2} b, \frac{\sigma^2}{2(1+\sigma)^2} b \right)$ and we obtain at most one cluster item c_g for which $s_{c_g} < \frac{\sigma^2}{4(1+\sigma)^2} b$. Again, small items are clustered by non-increasing order of p_i/s_i. The algorithm continues analogously to case 1: We set $k := \left\lceil \log_{1+\sigma} \frac{4(1+\sigma)^2}{\sigma^2} \right\rceil$, which implies $(1+\sigma)^k \frac{\sigma^2}{4(1+\sigma)^2} \geq 1$. Furthermore, since $\sigma < 1$, we have $k \geq 2$. We define the following intervals and values:

  - $I_h := \left[(1+\sigma)^{h-1} \frac{\sigma^2}{4(1+\sigma)^2} b, (1+\sigma)^h \frac{\sigma^2}{4(1+\sigma)^2} b \right) \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{h-1} \frac{\sigma^2}{4(1+\sigma)^2} b \right\rceil, \left\lceil (1+\sigma)^h \frac{\sigma^2}{4(1+\sigma)^2} b \right\rceil - 1 \right]$ for all $1 \leq h \leq k-1$
  - $I_k := \left[(1+\sigma)^{k-1} \frac{\sigma^2}{4(1+\sigma)^2} b, (1+\sigma)^k \frac{\sigma^2}{4(1+\sigma)^2} b \right] \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{k-1} \frac{\sigma^2}{4(1+\sigma)^2} b \right\rceil, \left\lfloor (1+\sigma)^k \frac{\sigma^2}{4(1+\sigma)^2} b \right\rfloor \right]$
  - $t_h := \left\lceil (1+\sigma)^{h-1} \frac{\sigma^2}{4(1+\sigma)^2} b \right\rceil$ for all $1 \leq h \leq k$

  Again, if $s_i \in I_h$ for some $i \in L \cup C$ and $1 \leq h \leq k$, set $s'_i := t_h$. If $s_{c_g} < \frac{\sigma^2}{4(1+\sigma)^2} b$, set $s'_{c_g} := s_{c_g}$. The new minimum quantity q' and the new capacity b' are defined as in case 1.

After applying the transformations from case 1 or case 2, respectively, we solve the instance consisting of the items in $L \cup C$, their respective profits p_i and adjusted sizes s'_i and m bins with minimum quantity q' and capacity b'. Let I denote the original instance of the problem and let I' denote the instance that we have just created. Let an arbitrary but fixed optimal solution to I be given. Let load(j) (for some $1 \leq j \leq m$) denote the sum of the sizes of the items that are packed into bin j in the optimal solution to I that we are considering and let load$'(j)$ be defined analogously for the solution to I' that we construct in the following.
According to our definition, an (α, β)-approximation must return infeasibility of the given instance if there is no feasible solution to the instance. However, since an empty packing is a feasible solution to every instance of MWBMMQ, infeasibility can never occur.
We now show that there is a feasible solution to I' that induces a (possibly infeasible) solution to I that is at least as good as an optimal solution to I: Given such a solution to I', we

have $\text{load}'(j) \in \{0\} \cup [q', b']$ for all bins j. Expanding the large items that are used in the solution to their original (instead of their rounded) sizes and replacing the cluster items by their associated small items increases the load in each of the bins to at most $(1+\sigma)b'$. By definition of q' and b', we obtain a packing where $\text{load}(j) \in \{0\} \cup \left[\frac{q}{(1+\sigma)^2}, (1+\sigma)^3 b\right] \subseteq \{0\} \cup \left[\frac{q}{1+\epsilon}, (1+\epsilon)b\right]$ for all bins, which yields the claim. In order to show the existence of such a solution to I', we consider each of the two cases of the algorithm separately:

- Case 1: $q \leq \frac{\sigma}{2} b$
  First of all, note that $s'_i \geq \frac{s_i}{1+\sigma} \geq \frac{\sigma}{2(1+\sigma)} b \geq \frac{q}{1+\sigma} \geq \frac{q}{(1+\sigma)^2}$ for all $i \in L$. Considering the integrality of s'_i, we obtain $s'_i \geq \left\lceil \frac{q}{(1+\sigma)^2} \right\rceil = q'$, i.e. one large item is sufficient to fulfill the minimum quantity constraint of a bin in I'.
  We now construct a feasible solution to I' as claimed: First of all, we pack the large items (with their adjusted sizes s'_i) into the same bins as in the fixed optimal solution to I that we are considering. We obtain a solution in which $\text{load}'(j) \in \{0\} \cup [q', b]$ for all bins j. If there are any bins without large items, we use the cluster items (if there are any) to fill these bins. We process the empty bins one by one and pack the cluster items one after another so that $\text{load}'(j) \in [b, (1+\sigma)b]$ until all cluster items have been packed or until there are no bins without large items and into which we can pack a cluster item without exceeding $(1+\sigma)b$ are left. We *always* schedule cluster items ordered by non-increasing profit densities p_{c_r}/s_{c_r}. If any unused cluster items and bins j with large items for which $\text{load}'(j) < b$ remain, we continue packing cluster items into these bins until $\text{load}'(j) \geq b$ and as long as this does not cause the load of any of the bins to exceed $(1+\sigma)b$. Note that every cluster item has size in $\left[\frac{\sigma}{2} b, \sigma b\right)$, so this is in fact possible. Due to the integrality of the item sizes we obtain $\text{load}'(j) \leq \left\lfloor (1+\sigma)^2 b \right\rfloor = b'$. We obtain one of the following cases:

  a) *All cluster items have been packed and all bins are feasible regarding q' and b'.*
  In this case we have in fact created a feasible solution to I'. Since all the large items that are packed in the optimal solution to I as well as all small items (represented by the respective cluster items) are packed, the value of the solution we have created is at least as good as an optimal solution to I.

  b) *All cluster items have been packed and some bins are infeasible regarding q' or b'.*
  Note that by construction at each point in time there is at most one bin that is infeasible and it can only be infeasible with respect to q', as the load never exceeds b'. Assume that such a bin j' exists once all cluster items have been packed.

If there is no other bin with strictly positive load, then the optimal solution to I that we have started with, did not contain any large items. Furthermore, as the sum of the sizes of the cluster items is smaller than q', the sum of the sizes of the corresponding small items must be smaller than q, so the optimal solution to I we have considered cannot contain any small items either. This implies that the optimal solution to I must have been empty. Thus, an empty packing is a feasible solution to I' which is as good as the optimal solution to I.

If there is another bin j with strictly positive load, then by construction it is feasible with respect to q' and we have $\text{load}'(j) \leq (1+\sigma)b$. Moving all items from bin j' to bin j yields $\text{load}'(j') = 0$ and $\text{load}'(j) < (1+\sigma)b + q' \leq (1+\sigma)b + q \leq (1+\sigma)b + \frac{\sigma}{2}b < (1+\sigma)^2 b$. Considering the integrality of $\text{load}'(j)$ we obtain $\text{load}'(j) \leq \left\lfloor (1+\sigma)^2 b \right\rfloor = b'$.

Again, the same large items that are packed in the optimal solution to I, that we are considering, as well as all small items are packed. Hence, the value of the solution we have created is at least as good as an optimal solution to I.

c) *Some cluster items could not be packed.*

There cannot be any bin j for which $\text{load}'(j) < b$: Assume that there is a bin j for which $\text{load}'(j) < b$ and let c_r denote the next cluster item that would be packed by the above procedure if there was enough space left. By construction we have that $\text{load}'(j) + s_{c_r} < b + \sigma b = (1+\sigma)b$. As we are packing cluster items as long as we can do so without the load on any of the bins exceeding $(1+\sigma)b$, the fact that c_r was not packed contradicts our assumption. So for all bins j we have $\text{load}'(j) \in [b, (1+\sigma)b]$, which implies feasibility of the solution regarding q' and b' (Once again, we take the integrality of $\text{load}'(j)$ and b' into account).

By construction, there are no empty bins left. In an optimal solution to I, $\text{load}(j) \leq b$ for all j. This implies that the sum of the sizes of the cluster items (and thus of the corresponding small items) in the solution we have created is at least the sum of the sizes of the small items in the optimal solution to I that we are considering. Recall that small items have been clustered by non-increasing order of p_i/s_i. Furthermore, the above construction of a solution to I' packs cluster items by non-increasing order of p_{c_r}/s_{c_r}. All in all, we get that the profit from cluster items in the solution to I' we have created is at least as good as the profit from small items in an optimal solution to I. By construction, the large items in both solutions are identical. So we have in fact created a feasible solution to I' that is at least as good as an optimal solution to I.

- Case 2: $q > \frac{\sigma}{2}b$

  Let $\text{load}_s(j)$ and $\text{load}_l(j)$ denote the sum of the sizes of small and large items, respectively, that are packed into bin j in the optimal solution to I that we are considering. The values $\text{load}'_s(j)$ and $\text{load}'_l(j)$ are defined analogously for the solution to I' that we are constructing. As in case 1, we begin constructing a feasible solution to I' by packing the large items into the same bins as in the optimal solution to I that we are considering. Hence, $\text{load}'_l(j) \geq \frac{\text{load}_l(j)}{1+\sigma}$ for all bins j. Afterwards we fill all bins j for which $\text{load}_s(j) > 0$ with cluster items, so that $\text{load}'_s(j) \in \left[\frac{\text{load}_s(j)}{1+\sigma} - \frac{\sigma^2}{2(1+\sigma)^2}b, \frac{\text{load}_s(j)}{1+\sigma}\right)$. Note that the length of the interval is $\frac{\sigma^2}{2(1+\sigma)^2}b$, which implies that all cluster items are smaller than the length of the interval. Thus, if there are enough cluster items available, it is always possible to find such a packing. We have that $\sum_{i=1}^{g} s_{c_i} \geq \sum_{i=1}^{n} \frac{s_i}{1+\sigma} \geq \sum_{j=1}^{m} \frac{\text{load}_s(j)}{1+\sigma}$. This implies that there are enough cluster items to fill the bins as described. We continue packing cluster items until $\text{load}'_s(j) \in \left[\text{load}_s(j), \text{load}_s(j) + \frac{\sigma^2}{2(1+\sigma)^2}b\right)$ for all bins j for which $\text{load}_s(j) > 0$ or until all cluster items have been packed. Once again, as the length of the interval exceeds the size of the cluster items, we can find such a packing. As in case 1, cluster items are always packed by non-increasing order of p_{c_r}/s_{c_r}.

  We now consider the feasibility of this solution regarding q' and b':

  If $\text{load}(j) = 0$ for some bin j, we have $\text{load}'(j) = 0$ by construction.

  If $\text{load}(j) > 0$, we obtain

  $\text{load}'(j) \in \left[\frac{\text{load}_l(j)}{1+\sigma} + \frac{\text{load}_s(j)}{1+\sigma} - \frac{\sigma^2}{2(1+\sigma)^2}b, \text{load}_l(j) + \text{load}_s(j) + \frac{\sigma^2}{2(1+\sigma)^2}b\right] =$

  $\left[\frac{\text{load}(j)}{1+\sigma} - \frac{\sigma^2}{2(1+\sigma)^2}b, \text{load}(j) + \frac{\sigma^2}{2(1+\sigma)^2}b\right] \subseteq \left[\frac{q}{1+\sigma} - \frac{\sigma^2}{2(1+\sigma)^2}b, b + \frac{\sigma^2}{2(1+\sigma)^2}b\right]$. We have that $\frac{q}{1+\sigma} - \frac{\sigma^2}{2(1+\sigma)^2}b > \frac{q}{1+\sigma} - \frac{\sigma}{(1+\sigma)^2}q = \frac{(1+\sigma)q}{(1+\sigma)^2} - \frac{\sigma q}{(1+\sigma)^2} = \frac{q}{(1+\sigma)^2}$ and

  $b + \frac{\sigma^2}{2(1+\sigma)^2}b < (1+\sigma)b$ (since $\frac{\sigma}{2(1+\sigma)^2} < 1$).

  Taking the integrality of $\text{load}'(j)$ into account we get that $\text{load}'(j) \in [q', b']$.

  The profit from large items in the solution to I' is the same as in the optimal solution to I. By construction, all cluster items have been packed or $\text{load}'_s(j) > \text{load}_s(j)$ for all j (or both). In the first case it is immediately clear that the solution to I' is at least as good as the optimal solution to I. The argumentation regarding (super-)optimality of the solution in the second case is the same as in "*c) Some cluster items could not be packed.*" above.

  All in all, we have shown that the solution we have created is in fact feasible with respect to I' and its value is at least as good as the value of an optimal solution to I. We have already seen that this solution to I' induces a bicriteria solution to I that fulfills the claim.

All the steps necessary for creating the instance I' based on the input instance I such as clustering items and rounding item sizes can clearly be done in polynomial time. Note that k only depends on ϵ, so for $\epsilon > 0$ fixed, k can also be assumed to be fixed. Then according to Theorem 2.3.1, I' can be solved in polynomial time. □

2.4 Conclusion

In this chapter we have given an overview of previous work regarding bin packing problems with minimum quantities. Furthermore, we have provided a dynamic program and a bicriteria approximation algorithm that are both polynomial for certain special cases of GBPMQ.
In particular, we have shown polynomial solvability of the following special case of GBPMQ:

- GBPMQ where the minimum quantities are uniform, the capacities are uniform, the profits do not depend on the bins the number of different item sizes is bounded by a constant (Theorem 2.3.1)

In addition, we have shown the existence of a polynomial-time $(1, 1+\epsilon)$-algorithm (where $\epsilon > 0$ is arbitrary but fixed) for the following special case:

- GBPMQ where the minimum quantities are uniform, the capacities are uniform, the profits do not depend on the bins and the number of different item sizes is bounded by a constant (Theorem 2.3.2)

Chapter 3

Scheduling problems

3.1 Introduction

Scheduling problems occur in all kinds of processes, where similar activities are carried out by a number of persons or machines, possibly in parallel, and the objective is to minimize the time needed. There is a variety of parameters by which a scheduling problem can be characterized. An overview of problem definitions, their computational complexity and algorithms can be found in [9]. Practically speaking, idle times of machines or personnel are usually to be avoided, since costs for maintenance and salaries, respectively, are incurred even during idle times. Hence, utilizing the available resources as efficiently as possible is usually desirable. This makes scheduling problems natural candidates for adding minimum quantities. Furthermore, technical restrictions might enforce minimum quantities. Scheduling problems with minimum quantities have already been considered in [21] and in [45]. We summarize the definitions and the most important results from these publications in the next section.

3.2 Basics and Definitions

In [45], minimum quantities are applied to scheduling problems where either the sum of completion times or the makespan (i.e. the maximum completion time) are minimized. The problems are defined as follows:

Definition 3.2.1. $Pq_j||C_{\max}$ and $Pq_j||\sum C_i$ are given by $n \in \mathbb{N}$ jobs with processing times $p_i \in \mathbb{N}$ for $i \in \{1,\dots,n\}$ and $m \in \mathbb{N}$ identical parallel machines with corresponding minimum quantities $q_j \in \mathbb{N}$ for $j \in \{1,\dots,m\}$. If all minimum quantities are identical, we denote the problems by $Pq||C_{\max}$ and $Pq||\sum C_i$, respectively. In order for a solution to be feasible, every

machine j that processes at least one job must process at least q_j units of work.
Let C_i denote the completion time of job i in a specific schedule.
Given an instance of $Pq_j||C_{\max}$, the task is to find a feasible solution that schedules the jobs on a subset of the machines in such a way that $C_{\max} := \max_{i=1}^{n} C_i$ is minimized. For $Pq_j||\sum C_i$ the objective function to be minimized is $\sum_{i=1}^{n} C_i$.

More generally, this can be considered an extension of the three-field notation [20]: Minimum quantities are a property of the machine environment that can be added to arbitrary scheduling problems. For example, we denote the scheduling problem consisting of a fixed number of m identical machines with minimum quantity q and where the makespan is supposed to be minimized by $Pmq||C_{\max}$. For $m = 2$ the problem is denoted by $P2q||C_{\max}$. In this thesis we consider the following machine environments:

- *Identical* parallel machines, where the processing time of each job i is given by an integer p_i as in Def. 3.2.1.
- *Uniform* parallel machines, where a *processing requirement* p_i is defined for every job i and for every machine j the *speed* of the machine s_j is given. The *processing time* of job i on machine j is $\frac{p_i}{s_j}$.
- *Unrelated* parallel machines, where the processing time of each job i on machine j is given by an integer $p_{i,j}$.

The three-field notation denotes the above machine environments by P, Q and R, respectively.

Remark 3.2.2. If a scheduling problem is defined on uniform machines in this thesis, minimum quantity constraints refer to the sum of *processing times* $\frac{p_i}{s_j}$ on the respective machines.

In the following we denote the *sum of processing times* of jobs that are scheduled on machine j by load(j). In particular, if the machine environment consists of uniform machines, load(j) refers to the sum of all $\frac{p_i}{s_j}$ for which job i is scheduled on machine j. We refer to a machine j as *active* if load(j) > 0, else we refer to it as *inactive*. If a machine processes a job *at a certain point in time*, we call it *busy* at that point in time, else we refer to it as *idle* at that point in time.
If there are no release dates, precedence constraints or other constraints that might cause idle times between the processing of jobs, then, unless specified otherwise, we assume that all jobs are processed consecutively on each machine without any idle times between two jobs, so that machine j becomes idle after load(j) time units.

In the following section we also consider the lateness minimization problem with minimum quantities as defined in [21]. We use the "delivery time model" that is also used in [21] in order to ensure that the objective function can only attain positive values so that our definition of approximation algorithms is well-defined for these problems.

Definition 3.2.3. $Pq_j||L_{\max}$ is given by $n \in \mathbb{N}$ jobs with processing times and delivery times $p_i, d_i \in \mathbb{N}$ for $i \in \{1,\dots,n\}$ and $m \in \mathbb{N}$ machines with corresponding minimum quantities $q_j \in \mathbb{N}$ for $j \in \{1,\dots,m\}$. We denote the lateness of a job i (i.e. the point in time at which a job or the output of the job is delivered) by $L_i := C_i + d_i$.
The task is to schedule the jobs on a subset of the machines in such a way that $L_{\max} := \max_{i=1}^{n} L_i$ is minimized. Every machine j that processes at least one job must process at least q_j units of work in order for the schedule to be feasible.
If all minimum quantities are identical, we denote the problem by $Pq||L_{\max}$.

We also generalize the idea of open shop problems. Let us first consider the following definition [9]:

Definition 3.2.4 (open shop scheduling)**.** There are n jobs and m machines. Every job i consists of a set of m operations $o_{i,j}$ with processing times $p_{i,j}$ for $i \in \{1,\dots,n\}$ and $j \in \{1,\dots,m\}$. Each job can only be processed by one machine at a time. Operation $o_{i,j}$ has to be processed on machine j. Given an objective function f that is nondecreasing regarding the completion times of the jobs, the problem is to find a feasible schedule that minimizes f. The problem is denoted by $O||f$.

Applying minimum quantities to this problem definition would not yield any interesting results, since the load that is processed by each machine is already determined by the respective instance. Thus, an instance would either be infeasible if the load assigned to at least one machine was less than the respective minimum quantity or the minimum quantities would not have any effect. Hence, we generalize the problem by assuming that there are m *groups* of parallel identical machines instead of m machines. Each operation $o_{i,j}$ must be processed on one of the machines in machine group j. Now it is no longer a priori clear how much load is scheduled on each of the machines and minimum quantity constraints become non-trivial. We now provide the formal definition of the problem:

Definition 3.2.5 (open shop scheduling with minimum quantities)**.** $Oq_j||f$ is given by $n \in \mathbb{N}$ jobs and $m \in \mathbb{N}$ machine *groups*. Each machine group j consists of $c_j \in \mathbb{N}$ identical machines with corresponding minimum quantities $q_j \in \mathbb{N}$. Every job i consists of a set of m operations $o_{i,j}$ with processing times $p_{i,j} \in \mathbb{N}$ for $i \in \{1,\dots,n\}$ and $j \in \{1,\dots,m\}$. Each job can only be processed by one machine at a time and operation $o_{i,j}$ must be processed by a machine in

machine group j. In order for a solution to be feasible, every machine that processes at least one job must process at least q_j units of work, where j denotes the machine group of the machine. Given an objective function f that is nondecreasing regarding the completion times of the jobs, the problem is to find a feasible schedule that minimizes f. The problem is denoted by $Oq_j||f$. If all minimum quantities are identical, we denote the problem by $Oq||f$.

We now briefly summarize the previous work on scheduling problems with minimum quantity constraints: First of all, note that $Pq||C_{\max}$ is strongly NP-hard, since this already holds for $P||C_{\max}$ [15] and $P||C_{\max}$ is the subproblem of $Pq||C_{\max}$ consisting of those instances for which $q = 0$. In addition, $P2||C_{\max}$ is weakly NP-hard, which implies that $P2q||C_{\max}$ and $Pmq||C_{\max}$ are also at least weakly NP-hard. On the other hand, $P||\sum C_i$ can be solved in polynomial time [21] and it is not immediately clear, whether this also holds for $Pq||\sum C_i$.
In [45] it is shown that, unless P=NP, $P2q||C_{\max}$ cannot be approximated with a ratio better than 2 in polynomial time. Obviously this also holds for $Pmq||C_{\max}$ and $Pq||C_{\max}$ and implies that there cannot be any PTAS for these problems. Assuming the existence of a $(1+\epsilon)$-approximation for $P||C_{\max}$ for some $\epsilon > 0$, it is shown how this algorithm can be turned into a $(1+\epsilon+\frac{1}{\beta}, \beta)$-approximation for $Pq||C_{\max}$, where $\beta \geq 1$ is an arbitrary upper bound on the ratio by which a minimum quantity constraint may be violated. Setting $\beta := 1$ and considering the existence of a $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithm for $P||C_{\max}$ and a FPTAS for $Pm||C_{\max}$, it is then concluded that a $(\frac{7}{3} - \frac{1}{3m})$-approximation for $Pq||C_{\max}$ and a $(2+\epsilon)$-approximation for $Pmq||C_{\max}$ exist [45]. In addition, a pseudo-polynomial algorithm for $Pmq||C_{\max}$ is given in [45], which implies *weak* NP-hardness of the problem. The analysis of makespan minimization problems in [45] is concluded by providing a polynomial-time algorithm for the case that preemption of jobs is allowed. With respect to minimizing the sum of completion times it is shown that $P2q||\sum C_i$ is (at least weakly) NP-hard.
In [21] it is shown that given a value $\epsilon > 0$, a $(2+\epsilon)$-approximation exists for $Pq|r_i|C_{\max}$, i.e. in the presence of release dates. Furthermore, the existence of a 4-approximation algorithm and a 5-approximation algorithm is shown for $Pq_j||C_{\max}$ and for $Pq_j|r_i|C_{\max}$, respectively. The pseudo-polynomial dynamic program for $Pmq||C_{\max}$ from [45] is extended in order to be applicable to $Pmq_j||C_{\max}$ as well. Moreover, a polynomial-time algorithm for $Pq_j|p_i = p|C_{\max}$ is provided. Minimum quantities are also applied to lateness minimization in [21]. It is shown that for every $\epsilon > 0$ there is a $(3+\epsilon)$-approximation algorithm for $Pq|d_i, r_i|L_{\max}$, a 5-approximation algorithm for $Pq_j|d_i|L_{\max}$ and a 6-approximation algorithm for $Pq_j|d_i, r_i|L_{\max}$.
In addition, [21] covers the minimization of the weighted sum of completion times. It is shown that there is an approximation algorithm for $Pq_j|r_i|\sum w_iC_i$, for which the approximation ratio depends on the number of machines. Moreover, a $(24 - \frac{12}{m})$-approximation is shown for $Pq_j||\sum w_iC_i$. The analysis of minimizing the sum of completion times is concluded by

proving that, unless P=NP, $P2q||\sum C_i$ cannot be approximated with a ratio better than $\frac{3}{2}$ in polynomial time, which implies that there cannot be any PTAS. Finally, a generic dynamic program for scheduling problems with minimum quantities is outlined in [21].

3.3 Complexity Results & Algorithms

We begin our analysis of scheduling problems with minimum quantities by generalizing one of the results from [45]. As we have already mentioned in the previous section, it is shown in [45] that if there is a $(1+\epsilon)$-approximation for $P||C_{\max}$ for $\epsilon > 0$, then there is a $(1+\epsilon+\frac{1}{\beta}, \beta)$-approximation for $Pq||C_{\max}$ for an arbitrary $\beta \geq 1$. We extend this result to the case where release dates are given and where the minimum quantities are not necessarily identical for all machines.

Theorem 3.3.1. *If there is a $(1+\epsilon)$-approximation for $P|r_i|C_{max}$ for some fixed $\epsilon > 0$, then there is also a $(1+\epsilon+\frac{1}{\beta}, \beta)$-approximation for $Pq_j|r_i|C_{max}$ for every fixed $\beta \geq 1$.*

Proof. Let an instance I of $Pq_j|r_i|C_{\max}$ be given and let OPT denote the makespan of an optimal solution to I. W.l.o.g. we assume that $\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} q_j$ for $m \geq 1$, else we iteratively remove the machine with the highest minimum quantity from the instance until the statement holds. If $\sum_{i=1}^{n} p_i < q_j$ for all j, then there is no feasible solution to the problem. By assumption there is a $(1+\epsilon)$-approximation for $P|r_i|C_{\max}$. For $k \in \{1,\dots,m\}$ we create instances I_k of $P|r_i|C_{\max}$, where the jobs and release dates are identical to those in I and there are k identical machines. Given some solution I_k, let S_j denote a schedule on machine j. If job i is contained in schedule S_j, we denote this by $i \in S_j$ in the following. We apply Algorithm 1. We claim that the algorithm is polynomial in the size of the input instance and that its output is a schedule that violates the minimum quantities by a factor at most β and for which the makespan is at most $(1+\epsilon+\frac{1}{\beta})$OPT.
First of all, we show that all steps of the algorithm are well-defined and that it finishes after a number of steps that is polynomially bounded in the size of the input instance. Note that S_1 is always assigned to machine 1 in line 11: We have that $\sum_{i \in S_1} p_i \geq \sum_{i \in S_j} p_i$ and $q_1 \leq q_j$ for all $j \geq 2$. If $\sum_{i \in S_1} p_i < q_1$ in some iteration $k \leq m$, this would contradict $\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} q_j$. If the algorithm refers to some machine j, then this machine always exists in the initial instance I, since by construction of the algorithm we have $j \leq d \leq k \leq m$ (except for line 21, where $d = k+1$ in order to stop the algorithm).

Algorithm 1 Reschedule $Pq_j|r_i|C_{\max}$

1: Input: Instances I_k for $k \in \{1,\dots,m\}$
2: **for** all $k = 1,\dots,m$ **do**
3: Apply the $(1+\epsilon)$-approximation for $P|r_i|C_{\max}$ to I_k, i.e. to an instance with k machines
4: Let t_k denote the makespan of the output schedule
5: Let S_d for $d \in \{1,\dots,k\}$ denote the schedule on machine d
6: Empty all machines and reorder them by non-decreasing q_j
7: Reorder the schedules S_j so that $\sum_{i \in S_{j_1}} p_i \geq \sum_{i \in S_{j_2}} p_i$ for $j_1 < j_2$
8: Set $j := 1$ and $d := 1$
9: **while** $d \leq k$ **do**
10: **if** $\sum_{i \in S_d} p_i \geq \frac{q_j}{\beta}$ **then**
11: Schedule S_d on machine j
12: Set $d := d+1$ and $j := j+1$
13: **else**
14: **if** $\left\{ d' \in \{d+1,\dots,k\} \middle| \sum_{r=d}^{d'} \sum_{i \in S_r} p_i \geq \frac{q_j}{\beta} \right\} \neq \emptyset$ **then**
15: Set $d' := \min \left\{ d' \in \{d+1,\dots,k\} \middle| \sum_{r=d}^{d'} \sum_{i \in S_r} p_i \geq \frac{q_j}{\beta} \right\}$
16: Schedule S_d on machine j
17: Schedule the jobs $i \in \bigcup_{r=d+1}^{d'} S_r$ in arbitrary order without idle times between two jobs on machine j starting at t_k
18: Set $d := d'+1$ and $j := j+1$
19: **else**
20: Schedule the jobs $i \in \bigcup_{r=d+1}^{k} S_r$ in arbitrary order without idle times between two jobs on machine 1 starting at t_k
21: Set $d := k+1$ and $j := j+1$
22: **end if**
23: **end if**
24: **end while**
25: **end for**
26: Output: Among the m schedules determine the one with the smallest makespan.

Note that for every machine j at most one of the blocks in the lines 11 - 12, in the lines 15 - 18 or in the lines 20 - 21 of the algorithm is executed once. Hence, the lines 11 and 16

are well-defined, because no jobs have been scheduled on the respective machine j before. We now show that line 17 is well-defined. Once line 17 is processed, schedule S_d for some $d \geq 2$ has been assigned to machine j in line 16. Since all schedules finish after at most t_k time units, machine j is idle after t_k. Hence, we can in fact schedule the jobs without idle times between two jobs after t_k on machine j.
The argument for showing that line 20 is well-defined is similar: We have already seen that S_1 is always scheduled on machine 1 in line 11. Afterwards j is increased and no other jobs are scheduled on machine 1 before line 20 is executed. Hence, machine 1 is idle after t_k and we can schedule the jobs on machine 1 without idle times between two jobs after t_k as assumed by the algorithm. For the remaining steps of the algorithm it is clear that they are well-defined.
We now provide a polynomial upper bound on the number of steps the algorithm requires. First of all, we consider the lines 10 - 23 of the algorithm: Every time, this part of the algorithm is executed, at most n jobs are scheduled, which requires at most $O(n)$ steps. Assuming that the sum of processing times in each of the schedules S_j has already been computed in line 7, the lines 14 and 15 can be executed in $O(m)$. The if-condition in line 10 can be evaluated in constant time $O(1)$. The same holds for increasing the variables d and j. So all in all, executing the lines 10 - 23 requires at most $O(m+n)$ steps. The while-loop starting in line 9 ends after at most m iterations. Therefore, the lines 9 - 24 require time at most $O(m^2+mn)$. By assumption, the approximation algorithm that is used in line 3 can be executed in polynomial time. Let T denote an upper bound on the number of steps required by the approximation algorithm. Executing the lines 4 - 8 and line 26 requires at most $O(n+m\log m)$ steps. Thus, the lines 3 - 24 can be executed using at most $O(m^2+mn+T)$ steps. The for-loop in line 2 is executed at most m times. This yields the overall bound $O(m^3+m^2n+mT)$ on the number of steps the algorithm requires. W.l.o.g. we can assume $m \leq n$, so the number of steps is also bounded by $O(n^3+nT)$.
We now show that the solution that the algorithm returns fulfills $\text{load}(j) \geq \frac{q_j}{\beta}$ for all active machines j: For machines that are processed in the lines 11, 16 and 17 of the algorithm this follows directly from the conditions in the lines 10 and 14, respectively. Machine 1 might be processed by the algorithm in line 20. If so, it is already feasible regarding q_1, because schedule S_1 has been scheduled on machine S_1 in line 11. Hence, $\text{load}(j) \geq \frac{q_j}{\beta}$ for all active machines j.
The output of an arbitrary iteration k of the algorithm is also feasible with respect to the release dates: All jobs that are scheduled during the time interval $[0,\dots,t_k]$ are scheduled according to one of the schedules S_d, which are feasible regarding the release dates. Note that $t_k \geq r_i$ for all i. Hence, the remaining jobs, which are scheduled after t_k, have already

been released.

We still have to show that the makespan of the resulting schedule is in fact at most $(1+\epsilon+\frac{1}{\beta})$OPT. For an arbitrary optimal solution to I let k' denote the number of active machines. Then the makespan of an optimal solution to $I_{k'}$ is also OPT. Hence, we have $t_{k'} \leq (1+\epsilon)$OPT. In order to show the claim it is sufficient to show that the algorithm outputs a schedule with makespan at most $(1+\epsilon+\frac{1}{\beta})$OPT in iteration $k = k'$. We consider three types of machines separately. Note that for each machine j that is active in the resulting schedule exactly one of the following cases is true:

- $j = 1$*:*
  Schedule S_1 is processed by machine 1 and finishes after at most $t_{k'} \leq (1+\epsilon)$OPT time units. It is possible that additional jobs for which the sum of processing times is less than $\frac{q_j}{\beta}$ are scheduled on machine 1 after $t_{k'}$ without idle times in line 20 of the algorithm. Note that q_j for any $j \leq k'$ is a lower bound on OPT: By definition of k' there is an optimal solution to I in which exactly k' machines are active. W.l.o.g. these are the machines with the k' smallest minimum quantities. Hence, all jobs on machine j are finished before $t_{k'} + \frac{q_j}{\beta} \leq (1+\epsilon+\frac{1}{\beta})$OPT.

- $j \geq 2$ *and jobs are scheduled on machine* j *in line 11 of the algorithm:*
  A schedule S_d for some d that finishes after at most $t_{k'} \leq (1+\epsilon)$OPT time units is scheduled on machine j.

- $j \geq 2$ *and jobs are scheduled on machine* j *in lthe lines 16 and 17 of the algorithm:*
  A schedule S_d for some d that finishes after at most $t_{k'} \leq (1+\epsilon)$OPT time units is scheduled on machine j. Afterwards all jobs from the schedules S_r for $r \in \{d+1, \ldots, d'\}$ are scheduled without idle times after $t_{k'}$. We claim that $\sum_{r=d+1}^{d'} \sum_{i \in S_r} p_i < \frac{q_j}{\beta}$. Assume that his is not the case. By construction we have $\sum_{i \in S_d} p_i \geq \sum_{i \in S_{d'}} p_i$. Hence, we also have $\sum_{r=d}^{d'-1} \sum_{i \in S_r} p_i \geq \frac{q_j}{\beta}$. If $d' = d+1$, then $\sum_{i \in S_d} p_i > \frac{q_j}{\beta}$, which contradicts the fact that machine j is not processed in line 11 of the algorithm. If $d' > d+1$, this contradicts the choice of d' in line 15. Hence we have that $\sum_{r=d+1}^{d'} \sum_{i \in S_r} p_i < \frac{q_j}{\beta}$. So all jobs on machine j have been completely processed before $t_{k'} + \frac{q_j}{\beta}$. As in the first case we argue that q_j for some $j \leq k'$ is a lower bound on OPT and thus all jobs on machine j are finished before $(1+\epsilon+\frac{1}{\beta})$OPT.

□

We obtain a similar result for $Pq_j||L_{\max}$:

Theorem 3.3.2. *If there is a polynomial-time* $(1+\epsilon)$*-approximation for* $P||L_{max}$ *for some* $\epsilon > 0$*, then there is also a polynomial-time* $(1+\epsilon+\frac{1}{\beta}, \beta)$*-approximation for* $Pq_j||L_{max}$ *for* $\epsilon > 0$ *for every fixed* $\beta \geq 1$.

Proof. We show the proof by reusing most of the algorithm and the argumentation from the proof of 3.3.1. In Algorithm 2 only those lines are depicted that differ from the respective lines of Algorithm 1. We denote the instance of $Pq_j||L_{\max}$ by I and create m instances I_k of $P||L_{\max}$ where k indicates the number of machines that is available in I_k. As before, we assume w.l.o.g. that $\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} q_j$. The number of steps required by the operations in Algorithm 2 that differ from those in Algorithm 1 is clearly polynomially bounded in the size of the I. So by the same argument as in the proof of Theorem 3.3.1 we get that that the number of steps required by Algorithm 2 is polynomially bounded in the size of I.
The lines 11 and 16 of the algorithm are well-defined, because no jobs have been scheduled on the respective machine j yet. The sum of processing times to be scheduled in the lines 17 und 20 of the algorithm is at most $\frac{q_j}{\beta}$. Regarding line 20 this is obvious from the algorithm. For line 17 this has been shown in the proof of Theorem 3.3.1. Machine j is idle before $\frac{q_j}{\beta}$, thus the lines 17 and 20 are also well-defined.
The claim that the minimum quantity constraints are violated by at most a factor β follows by the same argument as in the proof of Theorem 3.3.1.
In order to show that the algorithm is in fact a $(1+\epsilon+\frac{1}{\beta}, \beta)$-approximation, we assume that there is an optimal solution to I with value OPT that uses k' active machines. Again, we have $t_{k'} \leq (1+\epsilon)$OPT because an optimal solution to $I_{k'}$ also has objective value OPT. We distinguish two cases:

- *A job is assigned in line 11 or in line 16:*
  By construction, the lateness of the jobs in S_d is at most $(1+\epsilon)$OPT if the schedule starts at 0. We have that $\frac{q_j}{\beta} \leq \frac{\text{OPT}}{\beta}$ for all $j \leq k'$, so shifting the schedule to starting time $\frac{q_j}{\beta}$ increases the lateness of the jobs to at most $(1+\epsilon+\frac{1}{\beta})$OPT.

- *A job is assigned in line 17 or in line 20:*
  We have already shown that jobs are scheduled in such a way by the lines 17 and 20 that they are completed before $\frac{q_j}{\beta}$. Hence, they are completed at most $\frac{q_j}{\beta}$ time units later than in an optimal solution and their lateness becomes at most $(1+\frac{1}{\beta})$OPT.

Algorithm 2 Reschedule $Pq_j||L_{\max}$

3: Apply the $(1+\epsilon)$-approximation for $P||L_{\max}$ to I_k

...

11: Schedule S_d on machine j starting at $\frac{q_j}{\beta}$

...

16: Schedule S_d on machine j starting at $\frac{q_j}{\beta}$

17: Schedule the jobs $i \in \bigcup_{r=d+1}^{d'} S_r$ in arbitrary order without idle times between two jobs on machine j starting at 0

...

20: Schedule the jobs $i \in \bigcup_{r=d+1}^{k} S_r$ in arbitrary order without idle times between two jobs on machine 1 starting at 0

□

Note that schedules S_d for $d \geq 2$ could also be scheduled starting at 0 in line 11. However, this would not improve the approximation ratio of the algorithm, so we have decided not to add another distinction of cases to the algorithm.
We obtain the following corollary:

Corollary 3.3.3. *For all $\epsilon > 0$ there is a polynomial-time $(2+\epsilon)$-approximation algorithm for $Pq_j|r_i|C_{max}$ and $Pq_j||L_{max}$ and a $(3+\epsilon)$-approximation algorithm for $Pq_j|r_i|L_{max}$.*

Proof. Let an arbitrary $\epsilon > 0$ be given. According to [22], there is a PTAS for $P|r_i|L_{\max}$. So in particular, for every fixed $\epsilon > 0$ there is a polynomial-time $(1+\epsilon)$-approximation algorithm for $P|r_i|C_{\max}$ (i.e. all $d_i = 0$) and for $P||L_{\max}$ (i.e. all $r_i = 0$). Applying Theorem 3.3.1 and Theorem 3.3.2 with $\beta := 1$, yields the claim that there is a polynomial-time $(2+\epsilon)$-approximation algorithm for $Pq_j|r_i|C_{\max}$ and $Pq_j||L_{\max}$ for every fixed $\epsilon > 0$.
Let an instance of $Pq_j|r_i|L_{\max}$ be given and let OPT denote its optimal value. If we relax the problem by omitting the release dates, it becomes an instance of $Pq_j||L_{\max}$ for which we can find a schedule with maximal lateness at most $(2+\epsilon)$OPT, as we have just seen. The schedule is feasible with respect to the minimum quantities but it might be infeasible with respect to one or more release dates. We shift the schedule so that it starts at $r := \max_{r=1}^{n} r_i$ in

order to ensure that all jobs have been released. Note that OPT $\geq r$, which implies that the maximal lateness increases to at most $(3+\epsilon)$OPT. □

Note that this is a significant improvement of the approximation ratios given in [21]. We now turn our attention to minimizing the sum of completion times and show a bound on the best fixed approximation ratio we can obtain in polynomial time:

Theorem 3.3.4. *Unless P=NP, $P2q||\sum C_i$ cannot be approximated with a ratio $2-\epsilon$ in polynomial time for any $\epsilon > 0$.*

Proof. Assume that there is a $(2-\epsilon)$-approximation for $P2q||\sum C_i$ for some $\epsilon > 0$. Let $k' := \left\lceil \frac{4}{\epsilon} \right\rceil$. We show that the approximation could be used to solve PARTITION in polynomial time, which is not possible unless P=NP.
Let an instance of PARTITION be given by n integers $p_i \geq 1$. Set $B := \frac{1}{2}\sum_{i=1}^{n} p_i$. We interpret these integers as the processing times of n jobs. We set $k := \max\{n, k'\}$ and create $2k$ jobs with processing time $B+1$. Note that $k \in O(n)$, since k' is fixed. We set $q := B + k(B+1)$. Note that the sum of all processing times is $2B + 2k(B+1) = 2q$. By construction, a feasible schedule that uses both machines exists if and only if there is a feasible solution to the instance of PARTITION that we are considering. Furthermore, such a schedule induces a solution to PARTITION. An optimal schedule on two machines is outlined in Figure 3.3.1. Note that in an optimal schedule the jobs must be processed in order of non-decreasing processing times on each machine. Thus, the jobs that correspond to integers from the instance of PARTITION are processed first. They are indicated by the grey rectangles. The dashed rectangle represents additional jobs of length $B+1$.

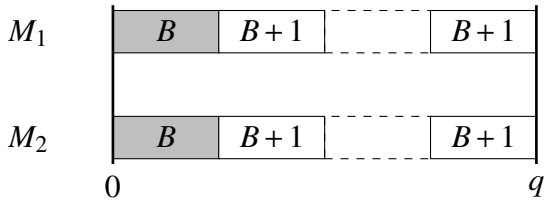

Fig. 3.3.1 An optimal solution on two machines

Analogously, an optimal solution on one machine is depicted in Figure 3.3.2.

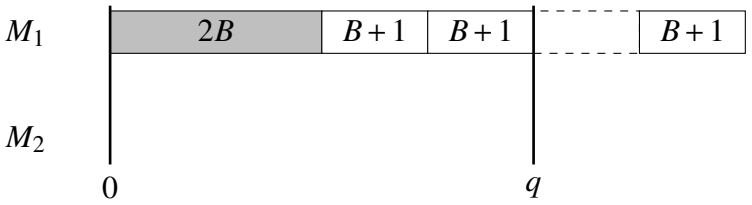

Fig. 3.3.2 An optimal solution on one machine

We now compare the optimal sum of completion times on one machine to the optimal sum of completion times on two machines. Note that we can calculate the sum of completion times of the jobs with length $B+1$ exactly. However, the sum of completion times of the jobs corresponding to the integers from the instance of PARTITION depends on the specific instance. We calculate a *lower* bound on the optimal sum of completion times on one machine, which we denote by OPT_1, and an *upper* bound on the optimal sum of completion times on two machines, which we denote by OPT_2. In order to calculate the lower bound on OPT_1, we only consider the completion times of the jobs with length $B+1$:

$$\mathrm{OPT}_1 \geq (B+1)\sum_{i=1}^{2k} i + 4Bk = (B+1)(2k^2+k) + 4Bk = 2Bk^2 + 5Bk + 2k^2 + k$$

The following reasoning leads to the above bound: If we simply schedule $2k$ jobs of length $B+1$ on one machine, then the first job has completion time $B+1$, the second job has completion time $2(B+1)$ and so on. This is reflected by the term $(B+1)\sum_{i=1}^{2k} i$. In the presence of the small jobs from the instance of PARTITION, the point in time at which the jobs with length $B+1$ start being processed is $2B$. This means that the completion time of each of the big jobs increases by $2B$ and there are $2k$ such jobs, which leads to the term $4Bk$. Analogously we determine an upper bound on OPT_2:

$$\mathrm{OPT}_2 \leq Bn + 2\left((B+1)\sum_{i=1}^{k} i + Bk\right) = Bn + (B+1)(k^2+k) + 2Bk \leq Bk^2 + 4Bk + k^2 + k$$

We now show correctness of the first inequality by constructing a feasible schedule for which this upper bound on the sum of completion times holds. Then this is obviously also an upper bound on OPT_2. Recall that in order for a feasible solution on two machines to exist, a solution to PARTITION must exist. Given a solution to PARTITION, we schedule the jobs corresponding to the integers of the first partition on the first machine and the jobs corresponding to the integers of the second partition on the second machine without idle times. This yields a schedule with n jobs that have completion times at most B. Hence,

the sum of completion times for these jobs is at most Bn. Processing k jobs of length $B+1$ beginning at B yields a sum of completion times $(B+1)\sum_{i=1}^{k} i + Bk$ for the respective jobs. We schedule k jobs of length $B+1$ on each of the two machines so that we have to consider this bound twice. Hence, we have established the first inequality. The last inequality follows from $k \geq n$.
We are now ready to prove a lower bound on the ratio by which the two values differ:

$$\frac{\mathrm{OPT}_1}{\mathrm{OPT}_2} \geq \frac{2Bk^2+5Bk+2k^2+k}{Bk^2+4Bk+k^2+k} = 2 - \frac{3Bk+k}{Bk^2+4Bk+k^2+k} > 2 - \frac{4Bk}{Bk^2} = 2 - \frac{4}{k} \geq 2-\epsilon$$

We have already stated that there is a feasible solution on two machines if and only of there is a solution to the instance of PARTITION. In this case, every solution on one machine is worse than an optimal schedule on two machines by a factor that exceeds $2-\epsilon$. Hence, a $(2-\epsilon)$-approximation algorithm would return a schedule on two machines and by doing so solve the given instance of PARTITION. Thus, a polynomial-time $(2-\epsilon)$-approximation algorithm cannot exist unless P=NP. □

We now show that there is in fact a 2-approximation algorithm for $Pq||\sum C_i$ which is the best possible approximation ratio we can obtain according to the previous theorem. In order to show the claim, we need the following result from [11]:

Lemma 3.3.5. *Let m' and m'' be two positive integers such that $m' < m''$. The sum of completion times of a set of jobs scheduled with the SPT list scheduling algorithm on m' machines is smaller than or equal to $\frac{m''}{m'}$ times the sum of completion times of the same jobs scheduled with the SPT list algorithm on m'' machines.*

Theorem 3.3.6. *There is a polynomial-time 2-approximation algorithm for $Pq_j||\sum C_i$.*

Proof. First of all, we show that if there is a feasible solution to a given instance of $Pq_j||\sum C_i$ with m' active machines for some $m' \leq m$, then the shortest processing time list scheduling algorithm (SPT, [19]) can be used to generate a schedule that is feasible regarding the minimum quantities if we apply the algorithm to the same instance where the number of machines is restricted to the $\left\lceil \frac{m'}{2} \right\rceil$ machines with the smallest minimum quantities.
More specifically, we run SPT for $\left\lceil \frac{m'}{2} \right\rceil$ machines, choose the $\left\lceil \frac{m'}{2} \right\rceil$ machines with the lowest minimum quantities from the instance of $Pq_j||\sum C_i$ and order them by non-decreasing minimum quantities. We assign the partial schedules on the individual machines returned by SPT to these $\left\lceil \frac{m'}{2} \right\rceil$ machines in such a way that $\mathrm{load}(j_1) \leq \mathrm{load}(j_2)$ for $j_1 < j_2$. We claim that this solution is feasible with respect to the minimum quantities. Set $q := q_{\left\lceil \frac{m'}{2} \right\rceil}$. Assume that there is a machine $j' \in \left\{1, \ldots, \left\lceil \frac{m'}{2} \right\rceil\right\}$ for which $\mathrm{load}(j') < q_{j'}$. If $j' = \left\lceil \frac{m'}{2} \right\rceil$, then the sum

of all processing times is strictly less than $q\left\lceil \frac{m'}{2} \right\rceil$, which contradicts the fact that there is a feasible solution on m' active machines, $\left\lceil \frac{m'}{2} \right\rceil$ of which have minimum quantity at least q. Hence, $j' < \left\lceil \frac{m'}{2} \right\rceil$. We remove the job that was scheduled last on each of the machines $\left\{j'+1, \ldots, \left\lceil \frac{m'}{2} \right\rceil\right\}$ and denote the set consisting of these jobs by J_1. After the removal of these jobs we have $\text{load}(j) \leq \text{load}(j') < q_{j'} \leq q_j$ for all $j \in \left\{j'+1, \ldots, \left\lceil \frac{m'}{2} \right\rceil\right\}$. The first inequality follows from the fact that if this was not the case, then SPT would have scheduled another job on machine j'. Unless $j' = 1$, there are machines $\{1, \ldots, j'-1\}$. By construction we have $\text{load}(j) \leq \text{load}(j') < q_{j'} \leq q$ for $j < j'$. We remove the jobs that are scheduled on the machines $\{1, \ldots, j'-1\}$ and denote the set consisting of these jobs by J_2. For the remaining schedule we have $\text{load}(j) \leq q_j$ for all $j \neq j'$ and $\text{load}(j') < q_{j'}$. This implies that every feasible schedule consisting of these jobs can contain at most $\left\lceil \frac{m'}{2} \right\rceil - 1$ active machines. In a feasible solution to the initial instance of $Pq_j \| \sum C_i$ on m' machines the jobs in J_1 can make at most $\left\lceil \frac{m'}{2} \right\rceil - j'$ of the machines $\left\{\left\lceil \frac{m'}{2} + 1 \right\rceil, \ldots, m'\right\}$ active. In addition, the jobs in J_2 can make at most $j'-1$ of the machines $\left\{\left\lceil \frac{m'}{2} + 1 \right\rceil, \ldots, m'\right\}$ active. So all in all, a feasible schedule can contain at most $\left\lceil \frac{m'}{2} \right\rceil - 1 + \left\lceil \frac{m'}{2} \right\rceil - j' + j' - 1 = 2\left\lceil \frac{m'}{2} \right\rceil - 2 < m'$ active machines, which contradicts our assumption. So the solution we have created using SPT on $\left\lceil \frac{m'}{2} \right\rceil$ machines must in fact be feasible.

If m' denotes the number of available machines, let $\text{SPT}^{m'}$ denote the value of a solution created by SPT list scheduling, let $\text{OPT}^{m'}$ denote the optimal sum of completion times *without* minimum quantities and let $\text{OPT}_{q_j}^{m'}$ denote the optimal sum of completion times *with* minimum quantities. Furthermore, let OPT denote the value of an optimal solution to the given instance of $Pq_j \| \sum C_i$. We obtain the following result:

$$\text{SPT}^{\left\lceil \frac{m'}{2} \right\rceil} \leq \frac{m'}{\left\lceil \frac{m'}{2} \right\rceil} \text{SPT}^{m'} \leq 2\text{SPT}^{m'} = 2\text{OPT}^{m'} \leq 2\text{OPT}_{q_j}^{m'} \tag{1}$$

The first inequality follows from Lemma 3.3.5 and the equality holds because SPT finds an optimal solution to $P \| \sum C_i$ [19]. The last inequality follows from the fact that the problem *without* minimum quantity constraints is a relaxation of the same problem *with* minimum quantity constraints.

W.l.o.g. we can assume that there is a feasible solution to the instance of $Pq_j \| \sum C_i$ that we are considering, since infeasibility of the instance can be checked easily. Then there is an optimal solution on m' *active* machines for some $m' \in \{1, \ldots, m\}$ and $\text{OPT}_{q_j}^{m'} = \text{OPT}$.

We have shown that SPT finds a schedule that is feasible regarding the minimum quantities if we restrict the number of machines to $\left\lceil \frac{m'}{2} \right\rceil$. Furthermore, we have seen in (1) that the sum of completion times of this schedule is at most twice the optimal value. Thus, applying

SPT to all possible numbers of machines $m' \in \{1,\dots,m\}$ and choosing the best among all feasible schedules is in fact a 2-approximation algorithm for $Pq_j||\sum C_i$. Since SPT is a polynomial-time algorithm, the number of times SPT is executed is polynomially bounded and selecting an optimal solution is possible in polynomial time, the algorithm we have presented is in fact a polynomial-time approximation algorithm. □

We now show that there is a polynomial-time bicriteria (α,β)-approximation algorithm for $Pq||C_{\max}$. Note that if we consider scheduling problems, the factor β refers to the minimum quantity constraints regarding the load of the machines. In order to prove the claim, we reuse a result from Chapter 2.

Theorem 3.3.7. *For every $\epsilon > 0$ there is a polynomial-time $(1+\epsilon, 1+\epsilon)$-approximation algorithm for $Pq||C_{max}$.*

Proof. Note that every instance of $Pq||C_{\max}$ can be interpreted as an instance of GBPMQ (Definition 2.2.4): For each machine we create a corresponding bin and for each of the n jobs we create an item i, the size s_i of which corresponds to the processing time of the respective job. We define unit profits for the jobs. We reuse the minimum quantity q and set the bin capacity to some value $b \geq q$. Then there is a feasible solution to the given instance of $Pq||C_{\max}$ with makespan at most b if and only if there is a feasible packing with profit n. Set $p := \sum_{i=1}^{n} p_i$. Clearly, for the optimal makespan OPT we have $\text{OPT} \in [q,p]$. Furthermore, the optimal objective value of the instance of GBPMQ that we have created is n for all $b \in [\text{OPT}, p]$.
In Theorem 2.3.2 a $(1,1+\epsilon)$-approximation algorithm for GBPMQ is given. If we apply this algorithm to the above instance of GBPMQ with $b \in [\text{OPT}, p]$, it returns a packing with value n and minimum quantities and bin capacities that are violated by at most a factor $(1+\epsilon)$. Let $\text{APP}(b)$ denote the value of the solution the bicriteria approximation algorithm returns if b is the bin capacity in the respective input instance.
We perform a binary search on $[q,p]$ in order to find the smallest value $b \in [q, \text{OPT}] \subseteq [q,p]$ for which $\text{APP}(b) = n$ where the search strategy is based on the assumption that $\text{APP}(b)$ is monotonously increasing in b. This assumption clearly holds for $b \in [\text{OPT}, p]$ where $\text{APP}(b) = n$. On the other hand, there might be $b_1 < b_2 < \text{OPT}$ so that $\text{APP}(b_2) < \text{APP}(b_1) \leq n$. However, the binary search still produces the desired output in this case: It starts with evaluating the endpoints of the interval $[q,p]$ where $\text{APP}(p) = n$. Then two cases are possible: During the binary search some value $b' < \text{OPT}$ with $\text{APP}(b') = n$ is found. In this case the binary search ends with some $b \leq b' < \text{OPT}$ with $\text{APP}(b) = n$. Otherwise, the binary search iteratively approaches OPT until it ends with $b = \text{OPT}$ and $\text{APP}(b) = n$.
Hence, we have found an approximate solution to GBPMQ that induces a corresponding

schedule with $\text{load}(j) \in \{0\} \cup \left[\frac{q}{(1+\epsilon)}, (1+\epsilon)b\right] \subseteq \{0\} \cup \left[\frac{q}{(1+\epsilon)}, (1+\epsilon)\text{OPT}\right]$ for all machines j. Note that $\text{APP}(b) = n$, i.e. all items are packed, which implies that all jobs are scheduled. Each execution of the approximation algorithm runs in polynomial time and the binary search finishes after at most $O(\log \sum_{i=1}^{n} p_i)$ iterations. So all in all, we can find an approximate solution in polynomial time. □

In fact, it is not always necessary to consider the complete interval $[q, p]$ when looking for an optimal makespan, as the following lemma shows:

Lemma 3.3.8. *Given a feasible instance of $Pq||C_{max}$, set $\alpha := \max\left\{\max_{i=1}^{n} p_i, \frac{1}{m}\sum_{i=1}^{n} p_i, q\right\}$. Let OPT denote the makespan of an optimal solution. Then OPT $\in [\alpha, 3\alpha]$.*

Proof. It is clear that α is a lower bound on OPT, so we only have to show that $\text{OPT} \leq 3\alpha$. Assume that there is a feasible instance of $Pq||C_{\max}$ for which $\text{OPT} > 3\alpha$ and let a corresponding solution be given. This implies that there is at least one machine j for which $\text{load}(j) = \text{OPT} > 3\alpha$. We now apply Algorithm 3.

Algorithm 3 Improve makespan

1: Input: An optimal schedule with makespan $\text{OPT} > 3\alpha$
2: **while** a machine j for which $\text{load}(j) = \text{OPT}$ exists **do**
3: Find jobs assigned to machine j so that their sum of processing times is in $[\alpha, 2\alpha)$. Denote the set of these jobs by J_j.
4: **if** an inactive machine j' exists **then**
5: Move the jobs in J_j from machine j to machine j' ▷ Then $\text{load}(j') \in [\alpha, 2\alpha)$ and $\text{load}(j) \in [\alpha, \text{OPT})$
6: **else**
7: **if** there is a machine $j'' \neq j$ with $\text{load}(j'') \leq 2\alpha$ **then**
8: Move one job from machine j to machine j'' ▷ Then $\text{load}(j'') \in [q, 3\alpha)$ and $\text{load}(j) \in [2\alpha, \text{OPT})$
9: **else**
10: STOP ▷ This contradicts $\alpha \geq \frac{1}{m}\sum_{i=1}^{n} s_i$
11: **end if**
12: **end if**
13: **end while**

We first assume that the algorithm reaches line 10 at some point: This implies that there is a machine j for which $\text{load}(j) = \text{OPT} > 3\alpha$ and all other machines are active and have load at least 2α. Hence, $\sum_{i=1}^{n} p_i > 2m\alpha$. However, this contradicts $\alpha \geq \frac{1}{m}\sum_{i=1}^{n} p_i$, which holds by

definition. Thus, the algorithm can never reach line 10.
Then each iteration of the while-loop decreases the number of machines for which load(j) = OPT by one. Thus, the algorithm terminates after at most m iterations of the while-loop and we have load(j) < OPT for all machines. Note that feasibility of the schedule is maintained throughout the algorithm, since $\alpha \geq q$. Hence, the algorithm returns a feasible schedule with a makespan that is strictly better than OPT. This contradicts optimality of the input schedule. Thus, OPT $\leq 3\alpha$. □

We can also use the idea of turning an instance of $Pq||C_{\max}$ into an instance GBPMQ in order to obtain the following result:

Theorem 3.3.9. *$Pq||C_{max}$ can be solved in polynomial time if the number of different processing times is bounded by some constant k.*

Proof. We transform a given instance of $Pq||C_{\max}$ into an instance of GBPMQ as described in the proof of Theorem 3.3.7 with bin capacity b, which is again a variable on which we perform a binary search. Note that the instance of GBPMQ that we have created meets the requirements for applying the dynamic program outlined in Theorem 2.3.1. This allows us to solve this instance in polynomial time. If the dynamic program returns a solution with value n for some bin capacity b, then this solution induces a schedule with makespan at most b. If the value of the solution is less than n, then there is no schedule with makespan at most b. As in the proof of Theorem 3.3.7, we perform a binary search on the interval of possible values the makespan can attain in order to find the smallest value b for which the dynamic program returns a packing with value n. Again, the binary search ends after at most $O(\log \sum_{i=1}^{n} p_i)$ steps so that we can in fact compute an optimal schedule in polynomial time. □

We now show that $Pq||\sum C_i$ can also be solved in polynomial time if the number of processing times is bounded by a constant.

Theorem 3.3.10. *$Pq||\sum C_i$ can be solved in polynomial time if the number of different processing times is bounded by a constant.*

Proof. We provide a dynamic program similar to the one given in Theorem 2.3.1 in order to solve the problem in polynomial time. Let the number of processing times be bounded by k. We group the jobs by their processing times and count the number of jobs in each group. For a group of jobs $c \in \{1, \ldots, k\}$ let p_c denote the corresponding processing time and let n_c denote the number of jobs in that group. Since k is an upper bound on the number of different processing times, there are at most k groups. We assume w.l.o.g. that there are exactly k such groups. Otherwise, we create empty dummy groups.

We denote the set of all possible sets of jobs by A, where each set of jobs is represented by a k-tuple:

$$A := \left\{ a \in \mathbb{Z}^k \middle| 0 \leq a_c \leq n_c \text{ for all } c \in \{1, \ldots, k\} \right\}$$

Furthermore, we define the set A_1 which consists of those tuples representing sets of jobs which are either empty or for which the sum of processing times is at least q, i.e. each element corresponds to a feasible assignment of jobs to one machine:

$$A_1 := \left\{ a \in A \middle| \sum_{c=1}^{k} a_c p_c \geq q \text{ or } a = 0 \right\}$$

Let $f(a, j)$ for some $a \in A$ and some $j \in \{1, \ldots, m\}$ denote the value of an optimal solution that schedules the jobs indicated by a in such a way that at most j machines are active. If there is no such solution, we use the convention $f(a, j) = \infty$. Let $\mathrm{OPT}(a)$ indicate the optimal sum of completion times if the jobs indicated by $a \in A$ are assigned to one machine. $\mathrm{OPT}(a)$ can be computed by scheduling the jobs in the order of non-decreasing processing times. We initialize f for all $a \in A$ as follows:

$$f(a, 1) := \begin{cases} \mathrm{OPT}(a) & \text{if } a \in A_1 \\ \infty & \text{else} \end{cases}$$

We iteratively increase j (until we reach $j = m$) and compute $f(a, j)$ for all possible $a \in A$ in each iteration as follows:

$$f(a, j) := \min_{a' \in A_1} \left(f(a - a', j - 1) + f(a', 1) \right)$$

Note that once we reach $j = m$, it is actually sufficient to compute $f(a, m)$ for $a = (n_1, \ldots, n_k)$, since all jobs have to be scheduled in order for the solution to be feasible.
We show by induction on j that the algorithm is correct, i.e. $f(a, j)$ is in fact the value of an optimal solution we obtain from scheduling the jobs corresponding to a on at most j machines: For the base case $j = 1$, the correctness of $f(a, 1)$ follows by construction.
For the induction step we assume that $f(a, j - 1)$ has been computed correctly for all $a \in A$. Given a set of jobs, the value of an optimal assignment of these jobs to j machines can then be determined by minimizing the sum of the value obtained from scheduling a subset of these jobs on $j - 1$ machines and the value obtained from scheduling the remaining jobs on one machine. This is exactly how $f(a, j)$ is defined.

Note that $f(a,j) \leq f(a,j-1)$, since 0 is contained in A_1. So once the above dynamic program has computed all the values $f(a,j)$, the optimal sum of completion times is given by $f(a,m)$ for $a = (n_1,\ldots,n_k)$.
We conclude the proof by showing that the number of steps required by the algorithm is polynomial in the size of the instance. The argumentation is similar to Theorem 2.3.1:

- Grouping the jobs by their processing times can be done in $O(n \log k)$ steps.
- For computing A_1, it is necessary to check for all possible combinations of jobs whether the corresponding sum of processing times is at least q. There are $\prod_{c=1}^{k}(n_c+1)$ combinations of jobs, so A_1 can be computed in $O(n^k)$. This implies that A_1 contains at most $\prod_{c=1}^{k}(n_c+1)$ elements.
- Assuming that all values required for the computation are already known, the computational complexity of determining $f(a,j)$ for one specific combination of a and j is $O(n^k)$. There are $(m-1)\prod_{c=1}^{k}(n_c+1)+1$ such computations of $f(a,j)$.
- So all in all, the number of steps required by the algorithm is bounded from above by $O(mn^{2k})$.

□

Remark 3.3.11. Note that the above dynamic program is very generic. We have used the fact that $Pq||\sum C_i$ is a minimization problem. By exchanging *min* for *max* in the calculation of $f(a,j)$ and ∞ for $-\infty$ in the initialization step of $f(a,1)$, this would also work for a maximization problem. Furthermore, the polynomial run time of the algorithm relies on the polynomial solvability of $P||\sum C_i$ on one machine. Note that we can neglect the minimum quantity here, since we filter out infeasible solutions during the initialization of $f(a,1)$. Furthermore, the dynamic program is based on the assumption that there are no interdependencies between jobs that are scheduled on different machines, i.e. once each job is assigned to a specific machine, we can optimize all machines individually. Note that this would not be the case in the presence of precedence constraints. The requirement to have a bounded number of processing times could be expressed more generally by requiring that the number of different values of each job property (such as processing time, weight, release date, etc.) is bounded. Then the number of possible combinations of these properties is also bounded. Finally, we require the machines to be identical. So all in all, the above dynamic program might work for all scheduling problems that fulfill the following properties (We say "might", since we have not given a formal definition of what a scheduling problem is in general and in order to exclude pathological cases.):

- There are n identical machines with minimum quantity q.
- The number of different values of each job property is bounded.
- The problem can be solved in polynomial time if there is only one machine.
- There are no interdependencies between jobs that are scheduled on different machines.

From the previous observation we obtain the following corollary:

Corollary 3.3.12. *$Pq||\sum w_iC_i$ can be solved in polynomial time if the number of different processing times and the number of different job weights are bounded.*

Proof. The first, the second and the fourth property that Remark 3.3.11 requires are fulfilled by assumption. The third property is fulfilled, since $1||\sum w_iC_i$ can be solved in polynomial time [46]. □

We now return to the general case of $Pq||\sum C_i$ with an arbitrary number of different processing times. In [45] it is shown that $P2q||\sum C_i$ (and thus $Pq||\sum C_i$) is at least weakly NP-hard. We now show that $Pq||\sum C_i$ is even strongly NP-hard:

Theorem 3.3.13. *$Pq||\sum C_i$ is strongly NP-hard.*

Proof. We show that there is a reduction from 3-PARTITION, which is known to be strongly NP-hard [16]. Let an instance of 3-PARTITION be given by $3n$ integers p_i and an integer B so that $\sum_{i=1}^{3n} p_i = Bn$ and $\frac{B}{4} < p_i < \frac{B}{2}$. The task is to partition these integers into n blocks where the sum of the integers in each block is B. Note that due to the sizes of the integers it is clear that if such a partitioning exists, then every block of the partition contains exactly three elements. Trivially, 3-PARTITION can be solved in polynomial time if n is fixed. Hence, we can assume w.l.o.g. that $n \geq 5$. We create an instance of $Pq||\sum C_i$ as follows: We create n machines with minimum quantity $q := 3Bn + B + 1$. For each of the integers p_i given by 3-PARTITION we create a job $i \in P := \{1, \ldots, 3n\}$ and interpret p_i as the corresponding processing time. In addition, we create n jobs $i \in A := \{3n + 1, \ldots, 4n\}$ with processing times $p_i := 3Bn + 1$. We claim that the following equivalence holds:
There is a feasible solution to the given instance of 3-PARTITION if and only if in every optimal solution to the instance of $Pq||\sum C_i$ all n machines are active (which implies that the load on each of these machines is $q = 3Bn + B + 1$).
Assume that in every optimal solution to the instance of $Pq||\sum C_i$ all n machines are active. As the set of feasible solutions is non-empty by construction, there is at least one such solution on n active machines. Note that in order for a machine to be active, at least one job

from A has to be scheduled on this machine. If this was not the case and there were only jobs from P scheduled on some machine j, then we would have $\text{load}(j) \leq Bn < 3Bn + B + 1 = q$. As there are n active machines and exactly n jobs in A, exactly one job from A is scheduled on each of the machines. In order for the solution to be feasible, the sum of processing times of the jobs from P on each machine has to be at least B. By construction, this is only possible if the sum of processing times of the jobs from P is *exactly* B on each machine. Obviously, this schedule induces a solution to 3-PARTITION.

We now show the other direction of the claim and assume that there is a feasible solution to the given instance of 3-PARTITION. First of all, note that in this case there is always a feasible schedule with n active machines, which is obtained by assigning all blocks of integers (i.e. of jobs in P) according to the solution to 3-PARTITION to different machines and adding one job from A to each machine. We now show that in this case a schedule with less than n active machines cannot be optimal. Let OPT denote the value of an optimal schedule and let $\text{OPT}_{<n}$ denote the value of an optimal solution if we restrict the number of available machines to at most $n-1$. We show that $\text{OPT} < \text{OPT}_{<n}$. We first establish an upper bound on OPT. As we have already seen, a solution on n machines schedules three jobs from P and one job from A on each machine. Furthermore, in order for the solution to be optimal, the jobs must be scheduled in the order of non-increasing processing times. The schedule on an arbitrary machine j is depicted in Figure 3.3.3, where the white rectangles represent jobs from P and the grey rectangle represents a job from A.

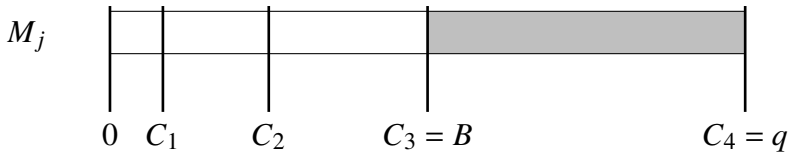

Fig. 3.3.3 The structure of a partial solution on one machine j

The actual sum of completion times depends on the processing times of the jobs in P, i.e. on the instance of 3-PARTITION.

For determining an upper bound on the sum of completion times, we solve the following LP. It is easy to see that the maximum is attained for $p_1 = p_2 = p_3 = \frac{B}{3}$.

$$\begin{aligned}
\text{maximize:} \quad & 3p_1 + 2p_2 + p_3 \\
\text{subject to:} \quad & \sum_{i=1}^{3} p_i = B \\
& p_i \leq p_{i+1} \qquad i = 1,2 \\
& p_i < \frac{B}{2} \qquad i = 1,2,3 \\
& p_i > \frac{B}{4} \qquad i = 1,2,3
\end{aligned}$$

Hence, we obtain the following upper bound on the sum of completion times on one machine: $\sum_{i=1}^{4} C_i \leq \frac{B}{3} + \frac{2B}{3} + B + 3Bn + B + 1 = 3Bn + 3B + 1$. Thus, OPT cannot exceed $3Bn^2 + 3Bn + n$.
We now consider $\text{OPT}_{<n}$. Given that there are at most $n-1$ machines and n jobs in A with processing time $3Bn+1$, at least one machine processes two jobs from A. We would like to derive a lower bound on $\text{OPT}_{<n}$, so we can ignore the completion times of the jobs in P and only focus on the completion times of the jobs from A. Every job $j \in A$ has completion time $C_j \geq 3Bn+1$ and there is at least one job $j \in A$ for which $C_j \geq 2(3Bn+1)$. Thus, we have that $\text{OPT}_{<n} \geq (n-1)(3Bn+1) + 2(3Bn+1) = 3Bn^2 + 3Bn + n + 1$. So all in all, we have $\text{OPT} \leq 3Bn^2 + 3Bn + n < 3Bn^2 + 3Bn + n + 1 \leq \text{OPT}_{<n}$. Hence, we have shown that if there is a feasible solution to the given instance of 3-PARTITION, then in every optimal solution to the instance of $Pq||\sum C_i$ all n machines must be active.
Thus, solving the instance of $Pq||\sum C_i$ either returns a solution on less than n active machines, i.e. there is no solution to 3-PARTITION, or it returns a schedule on n active machines which induces a solution to 3-PARTITION. This shows the claim. □

We now turn our attention to scheduling problems with minimum quantities on unrelated machines, in particular $Rq||C_{\max}$ and $Rq||\sum C_i$. Note that for unrelated machines the processing time of job i on machine j is given by some $p_{i,j}$. More information on scheduling problems on unrelated machines can be found in [9]. Note that we already know that $Rq||C_{\max}$ and $Rq||\sum C_i$ are strongly NP-hard because $P||C_{\max}$ and $Pq||\sum C_i$ are both strongly NP-hard ([16] and Theorem 3.3.13, respectively). For $P||C_{\max}$ and $Pq||\sum C_i$ there are polynomial-time fixed-ratio approximations ([45] and Theorem 3.3.6, respectively). Thus, we are interested in finding fixed-ratio approximations for $Rq||C_{\max}$ and $Rq||\sum C_i$ as well. However, we now show that, unless P=NP, such approximations cannot exist. In order to show the inapproximability, we require the following definition and remark:

Definition 3.3.14 (3-bounded 3-dimensional matching problem (3DM-3) [33])**.** Let X, Y, Z be sets with $|X| = |Y| = |Z| = n$. Let $T \subseteq X \times Y \times Z$ and each element of $X \cup Y \cup Z$ appears at most three times as a coordinate of T. The 3-bounded 3-dimensional matching problem is to find a maximum cardinality subset $M \subseteq T$ so that no two elements agree in any coordinate.

Remark 3.3.15. In [33], several complexity results due to [27] and [40] are summarized. The most important one in our context is the fact that there is an $\epsilon > 0$ for which it is NP-hard to decide, whether a given instance of 3DM-3 has a maximum matching of size n or whether all matchings have size at most $n(1-\epsilon)$ (hard gap at location 1).

We are now ready to show the claim:

Theorem 3.3.16. *For $\alpha > 1$ fixed, there cannot be any polynomial-time α-approximation algorithm for $Rq||C_{max}$ and $Rq||\sum C_i$, unless P=NP.*

Proof. We show the claim using a reduction from 3DM-3. Let an instance of 3DM-3 as described in Definition 3.3.14 be given. Let $m := |T|$. For each element in T we create a corresponding machine with minimum quantity $q = 3$ and for each element in $X \cup Y \cup Z$ we create a corresponding job. We obtain m machines and $3n$ jobs. For some machine j we denote the corresponding element in T by $T_j = \{x_j, y_j, z_j\}$. If some job i corresponds to one of the coordinates of T_j, we denote this by $i \in T_j$. For now, let k be a variable integer. We define the processing times as follows:

$$p_{i,j} := \begin{cases} 1 & \text{if } i \in T_j \\ kn & \text{else} \end{cases}$$

Assume that $M \subseteq T$ is a feasible solution to 3DM-3 for which $|M| = n$. We assign job i to machine j if $i \in T_j$. Obviously, we obtain a feasible schedule with value $3n$ (for $Rq||C_{\max}$) or $6n$ (for $Rq||\sum C_i$), respectively. Now assume that there is no feasible solution to 3DM-3 for which $|M| = n$: Then in every feasible schedule there is at least one job i that is assigned to a machine j so that $p_{i,j} = kn$. Thus, the ratio of the value of a feasible schedule if $|M| = n$ to the value of a feasible schedule if $|M| < n$ is at least $\frac{k}{3}$ (for $Rq||C_{\max}$) or $\frac{k}{6}$ (for $Rq||\sum C_i$), respectively. Now assume that there is an α-approximation for one of the problems. If we set $k := 6\alpha + 1$, the approximation algorithm must return a schedule in which all jobs i are assigned to machines j so that $p_{i,j} = 1$ if and only if there is a feasible solution to 3DM-3 for which $|M| = n$. In particular, this would allow us to determine in polynomial time if there is a solution with value n to a given instance of 3DM-3, which is NP-hard according to Remark 3.3.15. □

Note that the above proof can be easily adapted to other scheduling problems on unrelated machines.
In [45] it is shown that $Pq|pmtn|C_{\max}$ can be solved in polynomial time. We give now give a more general result.

Theorem 3.3.17. *Let $Pq|pmtn|f(C_i)$ be a scheduling problem with minimum quantities where $f(C_i)$ is an arbitrary objective function that only depends on the completion times of the jobs (In particular, this implies regular objective functions). If $P|pmtn|f(C_i)$ can be solved in polynomial time, then $Pq|pmtn|f(C_i)$ can also be solved in polynomial time.*

In order to prove the claim, we show that we can first solve the problem without minimum quantities and then reschedule the jobs in order to fulfill the minimum quantities. Let an instance of $Pq|pmtn|f(C_i)$ be given. As usual, n denotes the number of jobs and m denotes the number of machines. Set $p \coloneqq \sum_{i=1}^{n} p_i$ and $m' \coloneqq \min\left\{m, \left\lfloor \frac{p}{q} \right\rfloor\right\}$. Trivially, m' is an upper bound on the number of active machines a solution to the problem instance can use. We now solve the given instance of $Pq|pmtn|f(C_i)$ *without* the minimum quantities, i.e. $P|pmtn|f(C_i)$, on m' machines. By assumption, this can be done in polynomial time. Let $C_{\max}$ denote the maximal completion time in that schedule. We determine all the different points in time t_k at which a job begins, ends or is preempted. We order these points in time in increasing order $0 = t_0 < t_1 < \cdots < t_r = C_{\max}$. Note that the number of different points in time at which jobs begin or end is bounded from above by $2n$. Furthermore, the number of preemptions is polynomially bounded in the size of the instance of $P|pmtn|f(C_i)$ (and thus, in the size of the instance of $Pq|pmtn|f(C_i)$) since the schedule is the output of a polynomial-time algorithm. Hence, r is polynomially bounded in the size of the input instance of $Pq|pmtn|f(C_i)$. Based on these points in time we define intervals $T_k \coloneqq [t_{k-1}, t_k]$ for all $k \in \{1, \ldots, r\}$. For each T_k, let J_k denote the set of jobs that are processed during this interval. Note that $|J_k| \leq m'$ for all $k \in \{1, \ldots, r\}$. The intervals T_k and the corresponding sets of jobs J_k are used as an input for Algorithm 4.
The algorithm works as follows: For some machine $j \in \{1, \ldots, m'\}$ we denote the set of time periods during which the machine is idle by I_j. The algorithm starts with m' inactive machines so that we have $I_j = [0, C_{\max}]$ for all $j \in \{1, \ldots, m'\}$ in the beginning. The algorithm successively schedules load $\frac{p}{m'}$ on each machine. In order to do so, the time intervals T_k during which some machine j is idle are identified. Among these time intervals an interval for which $|J_k|$ is maximal is selected. This is necessary in order for the algorithm to be well-defined, as we will see in the proof of the claim. A fraction of an arbitrary job from J_k is selected and scheduled on machine j. Subsequently, I_j and J_k are updated. If we still have load$(j) < \frac{p}{m'}$ after scheduling the job, a job from another time interval T_k during which

machine j is idle and for which $|J_k|$ is maximal is scheduled. This procedure is repeated until $\text{load}(j) = \frac{p}{m'}$. If processing the selected job on machine j during the complete time interval T_k would cause $\text{load}(j)$ to exceed $\frac{p}{m'}$, the interval T_k is split. Therefore, we have to differentiate two cases (lines 5-9 and lines 10-15 of the algorithm). Once $\text{load}(j) = \frac{p}{m'}$, the algorithm processes machine $j+1$ or terminates if all machines have been processed.

Algorithm 4 Reschedule $Pq|pmtn|f(C_i)$

1: Set $I_j := [0, C_{\max}]$ for all $j \in \{1, \dots, m'\}$
2: Set $j := 1$ ▷ The machine that is currently processed by the algorithm
3: **while** $j \leq m'$ **do**
4: Among the time intervals for which $I_j \cap T_k = T_k$, select one for which $|J_k|$ is maximal
5: **if** $t_k - t_{k-1} > \frac{p}{m'} - \text{load}(j)$ **then**
6: Split T_k into $T_{k_1} := \left[t_{k-1}, t_{k-1} + \frac{q}{m'} - \text{load}(j)\right]$ and $T_{k_2} := \left[t_{k-1} + \frac{q}{m'} - \text{load}(j), t_k\right]$
7: Schedule one job i from J_{k_1} on machine j during T_{k_1}
8: Set $I_j := I_j \setminus T_{k_1}$ and $J_{k_1} := J_{k_1} \setminus \{i\}$
9: Set $j := j + 1$
10: **else**
11: **if** $t_k - t_{k-1} = \frac{p}{m'} - \text{load}(j)$ **then** ▷ The condition implies that we have $\text{load}(j) = \frac{p}{m'}$ after scheduling job i in line 14
12: Set $j := j + 1$
13: **end if**
14: Schedule one job i from J_k on machine j during T_k
15: Set $I_j := I_j \setminus T_k$ and $J_k := J_k \setminus \{i\}$
16: **end if**
17: **end while**

We need two auxiliary results:

Lemma 3.3.18. *Once the execution of the if-block (lines 5 - 16) starts, we always have $I_j \cap T_k = T_k$ or $I_j \cap T_k = \emptyset$ for all j and all k.*

Proof. We show the claim by induction on the number of times the if-block is executed. Let us consider the first execution of the if-block as the base case. Since $I_j := [0, C_{\max}]$ for all $j \in \{1, \dots, m'\}$ and $T_k \subseteq [0, C_{\max}]$ for all k, it is clear that $I_j \cap T_k = T_k$ for all j and all k.
For the induction step, we assume that the statement is true before the if-block is executed in some iteration. If the if-case is true, then the interval T_k is split and we get that $I_j \cap T_{k_1} = \emptyset$ and $I_j \cap T_{k_2} = T_{k_2}$. If the else-case applies, we get that $I_j \cap T_k = \emptyset$ at the end of the if-block. Nothing changes regarding the other intervals $T_{k'}$ with $k \neq k'$. Furthermore, the intervals T_k

and I_j do not change before the next start of the if-block, so the statement also holds in the next iteration. □

Lemma 3.3.19. *Once machines* $1,\dots,j$ *have been completely processed, we have that* $\max_k |J_k| \leq m' - j$.

Proof. Once again, we show this claim by induction on j. For $j = 0$, i.e. before the algorithm starts, this statement obviously holds by construction. For the induction step, we assume that the statement holds after the previous iteration in which machine j was processed, i.e. $\max_k |J_k| \leq m' - j$. If $\max_k |J_k| \leq m' - j - 1$, then the claim also holds for machine $j+1$, since $|J_k|$ never increases during the execution of the algorithm. Else, we have that $\max_k |J_k| = m' - j$ after iteration j. Recall that the algorithm assigns load $\frac{p}{m'}$ to each machine. Thus, load $\frac{jp}{m'}$ has already been assigned by the algorithm once iteration $j+1$ starts and the remaining load still to be assigned is $(m'-j)\frac{p}{m'}$. Now assume that $\sum_{k:|J_k|=m'-j} (t_k - t_{k-1}) > \frac{p}{m'}$, i.e. the sum of the lengths of the intervals T_k, for which $|J_k|$ attains the maximum, exceeds $\frac{p}{m'}$ at the beginning of iteration $j+1$. This yields the following contradiction, where the left-hand side of the inequality is a lower bound on the unscheduled load according to the assumption and the right-hand side is the actual unscheduled load at the beginning of iteration $j+1$:

$$(m'-j) \sum_{k:|J_k|=m'-j} (t_k - t_{k-1}) > (m'-j)\frac{p}{m'}$$

Hence, we have that $\sum_{k:|J_k|=m'-j} (t_k - t_{k-1}) \leq \frac{p}{m'}$. Once the algorithm processes machine $j+1$, the algorithm schedules load $\frac{p}{m'}$ from those time intervals T_k for which $|J_k|$ is maximal, i.e. $|J_k| = m' - j$. Since the sum of the lengths of these time intervals is at most $\frac{p}{m'}$, one job is removed from all these sets $|J_k|$ so that $\max_k |J_k|$ drops to $m' - j - 1$, which shows the claim. □

We are now ready to show the correctness of Theorem 3.3.17:

Proof of Theorem 3.3.17. First of all, we show that all steps of the algorithm are in fact well-defined. This is the case if an interval T_k for which $I_j \cap T_k = T_k$ and $|J_k| \geq 1$ always exists in line 4 of the algorithm .
Assume that this is not the case: If $I_j \cap T_k \neq T_k$ for all k, then according to Lemma 3.3.18 we have $I_j \cap T_k = \emptyset$ for all k. Note that $\bigcup_k T_k = [0, C_{\max}]$. If $I_j \cap T_k = \emptyset$ for all k, then machine j is busy during the whole interval $[0, C_{\max}]$. Since $C_{\max} \geq \frac{p}{m'}$ and the algorithm schedules at most load $\frac{p}{m'}$ on each machine and continues with machine $j+1$ afterwards or terminates, $I_j \cap T_k = \emptyset$ for all k is not possible. So, by Lemma 3.3.18, there is at least one k for which

$I_j \cap T_k = T_k$.

We assume that $|J_k| = 0$ holds for all k for which $I_j \cap T_k = T_k$. This implies that $|J_k| > 0$ only for those time intervals, for which $I_j \cap T_k = \emptyset$. By construction, these are exactly the time intervals during which machine j is busy. We have that $\text{load}(j) < \frac{p}{m'}$, because otherwise the algorithm would already have moved to machine $j+1$. Hence, once the algorithm finished processing machine $j-1$, the sum of the lengths of the intervals T_k for which $|J_k| \geq 1$ was at most $\text{load}(j) < \frac{p}{m'}$. Furthermore, according to Lemma 3.3.19, we had $\max_k |J_k| \leq m' - j + 1$. So once the algorithm had scheduled load $(j-1)\frac{p}{m'}$ on the first $j-1$ machines, the remaining load still to be assigned was at most $(m' - j + 1)\text{load}(j) < (m' - j + 1)\frac{p}{m'}$. This contradicts the fact that in total there is load p to be scheduled. Thus, in line 4 of the algorithm some k for which $I_j \cap T_k = T_k$ and $|J_k| \geq 1$ must always exist. Hence, the algorithm is in fact well-defined. The algorithm schedules load $\frac{p}{m'} \geq q$ on each machine, so the schedule we obtain is in fact feasible regarding the minimum quantities.

We now consider optimality of the solution created by Algorithm 4: By construction, m' is an upper bound on the number of machines that can be active in a feasible solution to the problem with minimum quantities. We use an optimal solution to the problem *without* minimum quantities on m' (not necessarily active) machines as an input to Algorithm 4. Trivially, the objective value of this solution is a lower bound on the optimal objective value of a solution to the initial instance of $Pq|pmtn|f(C_i)$. Algorithm 4 maintains the completion times of the jobs, which implies that it also maintains the objective value of the solution. Hence, the output of Algorithm 4 is in fact an optimal solution to the instance of $Pq|pmtn|f(C_i)$.

We now consider the number of steps required in order to compute an optimal solution to $Pq|pmtn|f(C_i)$: By assumption, $P|pmtn|f(C_i)$ can be solved in polynomial time. Furthermore, as we have already noted, r is polynomially bounded in the size of the input instance of $Pq|pmtn|f(C_i)$. This implies that the intervals T_k and the corresponding sets J_k can also be created in polynomial time. Once these inputs have been computed, Algorithm 4 is applied: Each iteration of the while-loop can be processed in polynomial time. The number of iterations of the while-loop *per machine* is bounded by the number of time intervals T_k because each interval is processed at most once for each machine. The algorithm starts with r time intervals. If a time interval is split, the number of time intervals increases by one. In every iteration in which a time interval is split we also set $j := j+1$. Thus, the number of time intervals at the end of the algorithm is bounded by $r + m'$ which implies that the number of iterations of the while-loop is bounded by $m'(r + m')$. Therefore the overall number of steps that is necessary for computing an optimal solution to $Pq|pmtn|f(C_i)$ is polynomially bounded. □

According to [37] $Q|pmtn|\sum C_i$ can be solved in polynomial time. Hence, this also holds for $P|pmtn|\sum C_i$ and using Theorem 3.3.17 we obtain the following result:

Corollary 3.3.20. *$Pq|pmtn|\sum C_i$ can be solved in polynomial time.*

Furthermore, according to [38], $P|pmtn|C_{\max}$ can be solved in polynomial time. Hence, we obtain the following corollary from Theorem 3.3.17:

Corollary 3.3.21. *$Pq|pmtn|C_{max}$ can be solved in polynomial time.*

Note that polynomial solvability of $Pq|pmtn|C_{\max}$ has already been shown in [45]. However, the definition of preemption in [45] differs from ours, as it is assumed that preemptions only take place at integral points in time, while we allow preemptions to take place at arbitrary points in time.
Having shown that many scheduling problems with minimum quantities that allow preemption stay polynomially solvable with minimum quantities, one might ask if this also holds for other machine environments, such as uniform machines. According to [37], $Q|pmtn;r_i|C_{\max}$ can be solved in polynomial time. However, we now show that the problem becomes NP-hard once we add minimum quantities:

Theorem 3.3.22. *$Qq|pmtn|C_{max}$ is NP-hard.*

Proof. We show the claim using a reduction from PARTITION. Let an instance of PARTITION be given by n integers p_i for $i \in \{1,\dots,n\}$ for which $\sum_{i=1}^{n} p_i = 2B$. We assume w.l.o.g that $p_i \leq B$ for all $i \in \{1,\dots,n\}$. We create an instance of $Qq|pmtn|C_{\max}$: For each of the integers p_i we create a corresponding job and a corresponding machine that both use the index i. The processing requirement of job i and the speed of machine i are both p_i, i.e., it takes one time unit to process job i on machine i. In addition, we create job $n+1$ with processing requirement $p_{n+1} \coloneqq B+1$ and machine $n+1$ with speed $2B+1$. Each of the machines has minimum quantity $q = 1$.
We claim that there is a feasible solution to PARTITION if and only if the optimal solution to the instance of $Qq|pmtn|C_{\max}$ is $C_{\max} = 1$.
Assume that there is a feasible solution to the instance of PARTITION that is given by P_1 and P_2 so that $P_1 \dot{\bigcup} P_2 = \{1,\dots,n\}$. Schedule all jobs $i \in P_1$ on the corresponding machine i. Schedule all jobs $i \in P_2$ as well as job $n+1$ on machine $n+1$. Recall that for uniform machines load(j) and the minimum quantity refer to the sum of all $\frac{p_i}{s_j}$ for which job i is scheduled on machine j. We now have load(i) $\in \{0,1\}$ for all machines $i \in \{1,\dots,n+1\}$ which is a feasible and optimal solution to $Qq|pmtn|C_{\max}$ of makespan 1.
Let an optimal solution to $Qq|pmtn|C_{\max}$ with $C_{\max} = 1$ be given. If machine $n+1$ was

inactive, then job $n+1$ would be processed with speed at most B on the machines $i \in \{1,\dots,n\}$. Since job $n+1$ can be processed only on one machine at each point in time and its processing requirement is $B+1$, its total processing time would be at least $\frac{B+1}{B} > 1$. This contradicts $C_{\max} = 1$. Therefore, machine $n+1$ must be active. For the minimum quantity of machine $n+1$ to be fulfilled and for load$(n+1)$ (and hence $C_{\max}$) not to exceed 1, the processing requirements on machine $n+1$ must be exactly $2B+1$. This leaves a sum of processing requirements B to be scheduled on the machines $i \in \{1,\dots,n\}$. Since $C_{\max} = q = 1$, all active machines must be busy during the time interval $[0,1]$. Hence, we have that $\frac{B}{\sum_{i\in P} p_i} = 1$ for some $P \subseteq \{1,\dots,n\}$. This implies that $\sum_{i\in P} p_i = B$, so we have found a solution to PARTITION.
Therefore, we can determine whether there is a solution to PARTITION by solving the instance of $Qq|pmtn|C_{\max}$. If so, the solution to $Qq|pmtn|C_{\max}$ induces a solution to PARTITION, which implies NP-hardness of $Qq|pmtn|C_{\max}$. □

Note that Theorem 3.3.22 implies that Theorem 3.3.17 cannot be extended to uniform machines.
We now consider open shop problems with minimum quantities as defined in Definition 3.2.5, in particular $Oq_j||C_{\max}$ and $Oq_j||\sum C_i$. Note that instances of $Oq_j||f$ for which $m_j = 1$ and $q_j = 0$ for all machine groups j are actually instances of $O||f$, where f denotes an arbitrary objective function. In addition, an instance of $Oq_j||f$ where $j = 1$, i.e. if there is only one machine group, is actually an instance of $Pq||f$. Hence, $Oq_j||f$ is at least as hard as $O||f$ and $Pq||f$. In particular, we obtain the following result:

Corollary 3.3.23. *$Oq_j||C_{max}$ and $Oq_j||\sum C_i$ are both strongly NP-hard and unless P=NP, there is no polynomial-time approximation with a fixed ratio better than 2.*

Proof. For $Oq_j||C_{\max}$ strong NP-hardness follows from the fact that $P||C_{\max}$ (and thus $Pq||C_{\max}$) is strongly NP-hard [15]. Inapproximability with a fixed ratio better than 2 is shown in [45] for $Pq||C_{\max}$.
In Theorem 3.3.13 we have shown that $Pq||\sum C_i$ is strongly NP-hard, so this also applies to $Oq_j||\sum C_i$. Inapproximability with a fixed ratio better than 2 follows from Theorem 3.3.4. □

In certain cases, open shop problems can be solved in polynomial time. We show that this is possible for unit size operations on the one hand and the case that preemption is allowed on the other hand. In order to do so, we require the following result from [18]:

Theorem 3.3.24. *Let an instance of $O|pmtn|C_{max}$ be given. Set $T_j := \sum_{i=1}^{n} p_{i,j}$ and $L_i := \sum_{j=1}^{m} p_{i,j}$. An optimal schedule can be computed in polynomial time and we have $C_{max} = \max\left\{\max_{j=1}^{m} T_j, \max_{i=1}^{n} L_i\right\}$.*

Theorem 3.3.25. *$Oq_j|pmtn|C_{max}$ can be solved in polynomial time.*

Proof. Let an instance I of $Oq_j|pmtn|C_{\max}$ be given and let c_j denote the number of machines in machine group j (cf. Definition 3.2.5). We show that we can solve $Oq_j|pmtn|C_{\max}$ in polynomial time in two steps:

- Step 1: Solving m instances of $Pq|pmtn|C_{\max}$
  Recall that every job i consists of j operations and operation $o_{i,j}$ must be processed by a machine in machine group j. For a fixed machine group j, we create an instance of $Pq|pmtn|C_{\max}$ by interpreting all operations $o_{i,j}$ as jobs that have to be processed on c_j identical machines with minimum quantity q_j. We denote this instance of $Pq|pmtn|C_{\max}$ by I_j. According to Corollary 3.3.21, $Pq|pmtn|C_{\max}$ can be solved in polynomial time. We apply a polynomial-time algorithm to all instances I_j for $j = 1,\dots,m$. Let OPT_j denote the optimal makespan of I_j.

- Step 2: Solving one instance of $O|pmtn|C_{\max}$
  Let the individual machines in machine group j be denoted by (j,k) for $k \in \{1,\dots,c_j\}$. We now split every operation $o_{i,j}$ into c_j operations: For each operation $o_{i,j,k}$ where $i \in \{1,\dots,n\}$, $j \in \{1,\dots,m\}$ and $k \in \{1,\dots,c_j\}$, $p_{i,j,k}$ is the processing time of job i on machine (j,k) in the solution to I_j that we have computed in step 1. Note that $p_{i,j,k} = 0$ is possible.
  We create an instance of $O|pmtn|C_{\max}$ consisting of the machines (j,k) for $j \in \{1,\dots,m\}$ and $k \in \{1,\dots,c_j\}$ as defined above (without minimum quantities) and n jobs. Each job i consists of the operations $o_{i,j,k}$ with processing times $p_{i,j,k}$ as defined above. For some machine (j,k), the operations $o_{i,j,k}$ for $i = 1,\dots,n$ are the ones that have to be processed by this machine. We denote this instance of $O|pmtn|C_{\max}$ by I'. According to Theorem 3.3.24 we can solve I' in polynomial time.

We claim that this solution to I' is actually an optimal (and feasible) solution to I, the initial instance of $Oq_j|pmtn|C_{\max}$. We show feasibility of the solution first: By construction, the load that is scheduled on each machine (j,k) in this solution to I' equals the load on the respective machine in the solution to I_j. Since this schedule was feasible with respect to the minimum quantities q_j, so is the solution to I'. Let $T'_{j,k}$ and L'_i be defined regarding I'

as in Theorem 3.3.24. Then, according to the theorem, the solution we have computed for I' has makespan $\mathrm{OPT}' = \max\left\{\max_{j=1}^{m}\max_{k=1}^{c_j} T'_{j,k}, \max_{i=1}^{n} L'_i\right\}$. We now consider the initial instance I. Let OPT denote the optimal makespan of I. Obviously $\max_{j=1}^{m}\mathrm{OPT}_j$ is a lower bound on OPT. By construction we have that $\max_{j=1}^{m}\max_{k=1}^{c_j} T'_{j,k} \leq \max_{j=1}^{m}\mathrm{OPT}_j$. Another lower bound on OPT is given by $\max_{i=1}^{n}\sum_{j=1}^{m} p_{i,j}$ and we have $\max_{i=1}^{n}\sum_{j=1}^{m} p_{i,j} = \max_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{c_j} p_{i,j,k} = \max_{i=1}^{n} L'_i$. Hence, we get that $\mathrm{OPT}' = \max\left\{\max_{j=1}^{m}\max_{k=1}^{c_j} T'_{j,k}, \max_{i=1}^{n} L'_i\right\} \leq \max\left\{\max_{j=1}^{m}\mathrm{OPT}_j, \max_{i=1}^{n}\sum_{j=1}^{m} p_{i,j}\right\} \leq \mathrm{OPT}$, which yields optimality of the solution. We find the optimal solution in polynomial time, as we can solve $Pq|pmtn|C_{\max}$ and $O|pmtn|C_{\max}$ in polynomial time according to Corollary 3.3.21 and Theorem 3.3.24. □

We conclude our analysis of open shop problems with the following theorem:

Theorem 3.3.26. *$Oq_j|p_{i,j} = 1|C_{max}$ can be solved in polynomial time.*

Proof. Let an instance of $Oq_j|p_{i,j} = 1|C_{\max}$ with n jobs and m machine groups be given. Then the sum of processing times that every machine has to process is n. As before, let c_j denote the number of machines in machine group j. Set $q := \max_{j=1}^{m} q_j$ and $m' := \min\left\{\left\lfloor\frac{n}{q}\right\rfloor, \min_{j=1}^{m} c_j\right\}$. In every feasible schedule there is at least one machine group that uses at most m' machines. Therefore, we have that $\left\lceil\frac{n}{m'}\right\rceil$ is a lower bound on $C_{\max}$. Furthermore, since the sum of the processing times corresponding to the (unit size) operations of each job is m and the operations of the same job cannot be processed concurrently, m is another lower bound on $C_{\max}$. We set $t := \max\left\{\left\lceil\frac{n}{m'}\right\rceil, m\right\}$ and show how to create a solution with optimal makespan $C_{\max} = t$.

We create a solution for machine group $j = 1$ on $m' \leq c_1$ machines. Schedule $q \geq q_1$ operations on each of the m' machines in such a way that none of the machines becomes idle before the last operation on the respective machine has been processed. By construction we have $m'q \leq n$. Hence, we can in fact schedule at least q operations on each machine. If there are any operations left, schedule them on arbitrary machines in such a way that $|\mathrm{load}(j_1) - \mathrm{load}(j_2)| \leq 1$ for all $j_1, j_2 \in \{1, \ldots, m'\}$. The resulting schedule on machine group $j = 1$ is obviously feasible and its makespan is $\left\lceil\frac{n}{m'}\right\rceil \leq t$. Based on this schedule we now create schedules for the machine groups $j = 2, \ldots, m$ using a simple permutation approach: If an operation $o_{i,1}$ is processed on the kth machine of machine group $j = 1$ for some $k \in \{1, \ldots, m'\}$, then it is also processed on the kth machine of all other machine groups. Hence, in order to define a schedule on all machine groups, it is sufficient to provide the point in time at which

the processing of each operation begins. Since $m' \geq c_j$ for all j, there are enough machines in each machine group in order for this to be well-defined. Furthermore, the sum of processing times on each machine is at least q so that such a schedule is feasible regarding all minimum quantities because $q \geq q_j$ for all j. Let $t_{i,j}$ denote the point in time at which the processing of operation $o_{i,j}$ starts. For $j = 1$ these values are already known and $t_{i,1} \leq t - 1$ for all i. Set $t_{i,j} \coloneqq t_{i,1} + j - 1 \pmod t$ for all $i \in \{1, \ldots, n\}$ and $j \in \{2, \ldots, m\}$. Hence, $t_{i,j} \leq t - 1$ for all i and all j and the completion time of all jobs is at most t, which implies optimality of the schedule. We still have to show that there is no point in time at which two operations $o_{i,j'}$ and $o_{i,j''}$ for $j' \neq j''$ are concurrently processed. By construction it is sufficient to show that $t_{i,j'} \neq t_{i,j''}$. Note that by construction $t_{i,j} = t_{i,1} + j - 1 \pmod t$ also holds for $j = 1$. For $j', j'' \in \{1, \ldots, m\}$ we have that $|j' - j''| \leq m - 1 \leq t - 1$. Hence $t_{i,1} + j' - 1 \pmod t = t_{i,1} + j'' - 1 \pmod t$ for $j' \neq j''$ is not possible. □

3.4 Conclusion

In this chapter we have given a summary of the existing research on the topic of scheduling problems with minimum quantities and we were able to contribute further results.
In particular, we have shown strong NP-hardness of the following problems:

- $Pq||\sum C_i$ (Theorem 3.3.13)
- $Oq_j||C_{\max}$ and unless P=NP, there is no polynomial-time approximation with a fixed ratio better than 2 (Corollary 3.3.23)
- $Oq_j||\sum C_i$ and unless P=NP, there is no polynomial-time approximation with a fixed ratio better than 2 (Corollary 3.3.23)

We could show (at least weak) NP-hardness of the following problem:

- $Qq|pmtn|C_{\max}$ (Theorem 3.3.22)

Furthermore, we have seen that there cannot be any polynomial-time fixed-ratio approximation algorithm for these problems, unless P=NP:

- $Rq||C_{\max}$ (Theorem 3.3.16)
- $Rq||\sum C_i$ (Theorem 3.3.16)

On the other hand, the following problems can be solved in polynomial time:

- $Pq||C_{\max}$ if the number of different processing times is bounded by a fixed integer k (Theorem 3.3.9)
- $Pq||\sum w_iC_i$ if the number of different processing times and the number of different job weights are bounded by fixed integers (Corollary 3.3.12)
- $Pq|pmtn|f(C_i)$ where $f(C_i)$ is an arbitrary objective function that only depends on the completion times of the jobs if $P|pmtn|f(C_i)$ can be solved in polynomial time (Theorem 3.3.17)
- $Pq|pmtn|\sum C_i$ (Corollary 3.3.20)
- $Pq|pmtn|C_{\max}$ (Corollary 3.3.21)
- $Pq||\sum C_i$ if the number of different processing times is bounded by a constant (Theorem 3.3.10)
- $Oq_j|pmtn|C_{\max}$ (Theorem 3.3.25)
- $Oq_j|p_{i,j}=1|C_{\max}$ (Theorem 3.3.26)

In addition, we were able to provide several polynomial-time approximations for NP-hard problems:

- If there is a $(1+\epsilon)$-approximation for $P|r_i|C_{\max}$ for some fixed $\epsilon>0$, then there is also a $(1+\epsilon+\frac{1}{\beta},\beta)$-approximation for $Pq_j|r_i|C_{\max}$ for every fixed $\beta\geq 1$ (Theorem 3.3.1)
- If there is a $(1+\epsilon)$-approximation for $P||L_{\max}$ for $\epsilon>0$, then there is also a $(1+\epsilon+\frac{1}{\beta},\beta)$-approximation for $Pq_j||L_{\max}$ for $\epsilon>0$ for every fixed $\beta\geq 1$ (Theorem 3.3.2)
- For every $\epsilon>0$ there is a polynomial-time $(2+\epsilon)$-approximation algorithm for $Pq_j|r_i|C_{\max}$ (Corollary 3.3.3)
- For every $\epsilon>0$ there is a polynomial-time $(2+\epsilon)$-approximation algorithm for $Pq_j||L_{\max}$ (Corollary 3.3.3)
- For every $\epsilon>0$ there is a polynomial-time $(3+\epsilon)$-approximation algorithm for $Pq_j|r_i|L_{\max}$ (Corollary 3.3.3)

- There is a polynomial-time 2-approximation algorithm for $Pq_j||\sum C_i$ (Theorem 3.3.6), which is the best possible fixed approximation ratio we can obtain in polynomial time (Theorem 3.3.4)
- For every $\epsilon > 0$ there is a polynomial-time $(1+\epsilon, 1+\epsilon)$-approximation algorithm for $Pq||C_{\max}$ (Theorem 3.3.7)

Chapter 4

Matching problems

4.1 Introduction

Matching problems are a broad area within combinatorial optimization and are historically closely tied to the development of polyhedral theory in the context of integer programming [41]. Some well-known examples of this class of problems are the maximum-weight matching problem and the perfect matching problem. We define matching problems with minimum quantities based on the maximum-weight b-matching problem. It consists of an undirected graph where each edge has a weight and each node v has a capacity b_v. The task is to find a maximum-weight multi-set of edges so that each node is matched by at most b_v edges [12]. We generalize this problem by adding a minimum quantity q_v on each node.

This model could be applied to the following example in intermediary trade: Assume that there is a set of suppliers and buyers of a certain commodity and a broker in between who wants to maximize his brokerage which depends linearly on the number of units sold. Each supplier has a limited supply and each buyer has a limited demand. Up to this point, this scenario could be modelled as an instance of the maximum-weight b-matching problem where the edge weights are uniformly one. Now assume that in addition some of the suppliers will only sell if a certain minimum number of units is sold. Analogously, some of the buyers will only buy if they receive at least a certain number of units. This might be motivated by transportation cost or administration overhead. This is where the minimum quantity constraints come into play that allow us to incorporate these additional requirements into the model.

A similar problem is defined in [2]. We provide the definitions from this article in the next section and use some of its results to complement our analysis.

Parts of this chapter have been presented at the conference OR 2018 in Brussels and the abstract has been published in the conference program [30].

4.2 Basics and Definitions

In this section we apply the idea of minimum quantities to the maximum-weight b-matching problem. The new problem definition we obtain is the basis for our analysis in the following section.

We use notation conventions that are commonly used in literature: Given a graph $G = (V, E)$, a matching vector $m \in \mathbb{N}^{|E|}$ and some edge $e = (v_1, v_2) \in E$ that connects the nodes v_1 and v_2, we denote the matching on this specific edge by m_e or by $m(v_1, v_2)$. For a set of edges $E' \subseteq E$ we use the notation $m(E') := \sum_{e \in E'} m_e$.

The maximum-weight b-matching problem is defined as follows:

Definition 4.2.1 (maximum-weight b-matching problem [12])**.** Let $G = (V, E)$ be an undirected graph and $b \in \mathbb{Z}^{|V|}$. A vector $x \in \mathbb{Z}^{|E|}$ is a b-matching if x is nonnegative and $x(\delta(v)) \leq b_v$ for all $v \in V$. Given a vector $w \in \mathbb{R}^{|E|}$, the maximum-weight b-matching problem is to find a b-matching x such that $w^T x$ is maximized.

A polynomial-time algorithm to solve the maximum-weight b-matching problem is given in [12] so that the following result is obtained:

Theorem 4.2.2 ([12])**.** *For any graph $G = (V, E)$, $b \in \mathbb{Z}^{|V|}$ and $w \in \mathbb{R}^{|E|}$, a maximum-weight b-matching can be found in strongly polynomial time.*

Furthermore, a variant of b-matchings with additional lower bounds on $x(\delta(v))$ is defined as follows:

Definition 4.2.3 ($[a', a'']$-matching [12])**.** Let $G = (V, E)$ be a graph, $a' \in \mathbb{Z}^{|V|}$, $a'' \in \mathbb{Z}^{|V|}$ and $w \in \mathbb{R}^{|E|}$. An $[a', a'']$-matching is a nonnegative vector $x \in \mathbb{Z}^{|E|}$ such that $a'_v \leq x(\delta(v)) \leq a''_v$ for all $v \in V$.

The problem of finding a feasible $[a', a'']$-matching with maximum weight is referred to as the maximum-weight $[a', a'']$-matching problem. Based on Theorem 4.2.2, the polynomial solvability of the maximum-weight $[a', a'']$-matching problem is derived in [12]:

Theorem 4.2.4 ([12])**.** *The maximum-weight $[a', a'']$-matching problem can be solved in strongly polynomial time.*

We now generalize Definition 4.2.1 by incorporating the concept of minimum quantities:

Definition 4.2.5 (maximum-weight b-matching problem with minimum quantities (MWB-MMQ))**.** Let $G = (V, E)$ be an undirected graph, $q_v \in \mathbb{N}$ and $b_v \in \mathbb{N}_+ \cup \{\infty\}$ for $v \in V$. A vector $m \in \mathbb{N}^{|E|}$ is a b-matching with minimum quantities q_v if for all $v \in V$ $q_v \leq m(\delta(v)) \leq b_v$ or $m(\delta(v)) = 0$. Given a vector $w \in \mathbb{Q}^{|E|}$, the maximum-weight b-matching problem with minimum quantities is to find a b-matching with minimum quantities that maximizes $w^T m$.

In addition, we define a generalization of MWBMMQ by allowing constraints on the *edges*.

Definition 4.2.6 (maximum-weight edge-constrained b-matching problem with minimum quantities (MWECBMMQ))**.** In addition to the properties and constraints given in Definition 4.2.5, let minimum quantities $q_e \in \mathbb{N}$ and capacities $b_e \in \mathbb{N}_+ \cup \{\infty\}$ for all $e \in E$ be given. In order for a matching to be feasible, we require $q_e \leq m_e \leq b_e$ or $m_e = 0$ for all $e \in E$.
We call this problem the maximum-weight edge-constrained b-matching problem with minimum quantities (MWECBMMQ).

Note that for an edge $e = (v_1, v_2)$ we can assume w.l.o.g. that $b_e \leq \max\{b_{v_1}, b_{v_2}\}$.
In the following, we mainly focus on MWBMMQ. However, considering MWECBMMQ will allow us to apply several results to flow problems in Chapter 5.
In Definition 4.2.5 we have adjusted the admissible range for some of the input values compared to Definition 4.2.1: As we require nonnegativity of x, we define b_v to be positive. If we had $b_v = 0$ for some $v \in V$, the node v and the edges incident to it could be deleted from the graph. Hence, w.l.o.g. we require b_v to be *strictly* positive. Additionally, as we evaluate the performance of algorithms with respect to the encoding size of a given problem instance, we restrict w to rational numbers so that there is a finite representation of each problem instance. A similar problem is defined in [2]:

Definition 4.2.7 (maximum-weight many-to-one matching problem with lower quotas (WMLQ) [2])**.** Let $G = (V, E)$ be an undirected, bipartite graph where $V = A \dot\cup P$ and $q, b \in \mathbb{N}^{|P|}$. A vector $m \in \{0,1\}^{|E|}$ is a feasible assignment if $q_v \leq m(\delta(v)) \leq b_v$ or $m(\delta(v)) = 0$ for all $v \in P$ and $m(\delta(v)) \leq 1$ for all $v \in A$. Given a vector $w \in \mathbb{Q}^{|E|}$, the maximum-weight many-to-one matching problem with lower quotas is to find a feasible assignment of maximum weight.
If $w_e = 1$ for all $e \in E$, the problem is referred to as MLQ.

The above definition is motivated by the problem of assigning applicants to posts where each applicant can be assigned to at most one post and the overall utility of the assignment is supposed to be maximized. The article [2] also generalizes the above definition by omitting the bipartiteness:

Definition 4.2.8 (generalized maximum-weight many-to-one matching problem with lower quotas (GWMLQ) [2])**.** Let $G = (V, E)$ be an undirected graph and $q, b \in \mathbb{N}^{|V|}$. A vector $m \in \{0,1\}^{|E|}$ is a feasible assignment if $q_v \leq m(\delta(v)) \leq b_v$ or $m(\delta(v)) = 0$ for all $v \in E$. Given a vector $w \in \mathbb{Q}^{|E|}$, the generalized maximum-weight many-to-one matching problem with lower quotas is to find a feasible assignment of maximum weight.
If $w_e = 1$ for all $e \in E$, the problem is referred to as GMLQ.

In [2] GWMLQ is shown to be solvable in polynomial time if $b_v \leq 2$ for all nodes v or if the problem is restricted to instances of bounded treewidth.

Note that the above definitions of WMLQ and GWMLQ differ from our definition of MWBMMQ, as they assume that each edge is used only *once*, while MWBMMQ allows a multi-set of edges to be selected. Still, there is a relationship between WMLQ and MWBMMQ, as the following remark shows:

Remark 4.2.9. Given an arbitrary instance of WMLQ, we can transform it into an equivalent instance of MWBMMQ by extending the vectors q and b, which are only defined for $v \in P$, to the nodes in A. We set $q_v := 0$ and $b_v := 1$ for all $v \in A$. Since each edge is incident to at least one node v for which $b_v = 1$, a feasible solution to this instance of MWBMMQ uses each edge at most once, despite MWBMMQ allowing a multi-set of edges to be selected. Thus, the sets of feasible solutions to these instances of WMLQ and MWBMMQ and the values of the respective solutions are the same, which implies that WMLQ is a special case of MWBMMQ.

Apart from [2], there are several other publications on matching problems with minimum quantities [3, 10, 26, 39]. The authors of these papers consider practical applications such as the assignment of residents to hospitals or the assignment of students to universities. In this regard they are similar to [2]. However, instead of assuming a numerical value for each individual assignment (where the goal is to maximize the overall value), these models assume that there are ordered lists of preferences. In some models only the preferences of the residents or students are considered, others also take the preferences of the hospitals or universities into account. Furthermore, some authors allow students to be assigned multiple times while others allow students to be assigned only once. The task is to find matchings that are Pareto optimal and strategy-proof. The authors of the above papers show that these matching problems are usually NP-hard. Since matching problems with ordinal preferences are not considered in this thesis, we do not elaborate on the results of the above papers in more detail.

Notation 4.2.10. Unless specified otherwise, we depict instances of MWBMMQ as graphs where the labels next to each node $v \in V$ specify q_v and b_v. Edges are labeled with their weights. If the edge weights are uniformly 1, we usually omit the edge labels.

In the following, unless specified otherwise, we assume that all graphs are simple, i.e. there are no loops or parallel edges. Note that if the graph contains parallel edges, then for MWBMMQ we obtain an equivalent instance by removing all edges except for the ones that maximize w_e for each set of parallel edges. This simplification step can be carried out in polynomial time.

4.3 Complexity Results & Algorithms

We now provide complexity results regarding MWBMMQ and present algorithms for solving or approximating MWBMMQ and variants of the problem efficiently. We begin by considering the complexity of MWBMMQ on general graphs. We prove its complexity by showing a reduction to the 3-bounded 3-dimensional matching problem (3DM-3, cf. Definition 3.3.14 and Remark 3.3.15).

Theorem 4.3.1. *Let $\epsilon > 0$ be a parameter for which it is strongly NP-hard to decide whether an instance of 3DM-3 has a solution with value n or whether all feasible solutions have value at most $(1-\epsilon)n$. Then this also holds with respect to MWBMMQ, even on bipartite graphs with $w_e = 1$ on all edges $e \in E$ and $q_v = b_v \leq 3$ for all $v \in V$.*

Proof. Let an instance of 3DM-3 be given as specified in Definition 3.3.14. We now create an instance of MWBMMQ as follows: For all elements $x_i \in X$, $y_i \in Y$ and $z_i \in Z$ $(i = 1,\ldots,n)$ we create nodes v_{x_i}, v_{y_i} and v_{z_i} and for each element $t_j \in T$ $(j = 1,\ldots,m)$ we create a node v_{t_j}. For each $t_j = (x_f, y_g, z_h)$ the edges (v_{t_j}, v_{x_f}), (v_{t_j}, v_{y_g}) and (v_{t_j}, v_{z_h}) are added. For all nodes v_{x_i}, v_{y_i} and v_{z_i} we set $q := b := 1$ and for all nodes v_{t_j} we set $q := b := 3$. Note that each v_{t_j} is incident to exactly three edges. The other nodes have degree at most m. An example for $n = 2$ and $m = 4$ is given in Figure 4.3.1 .

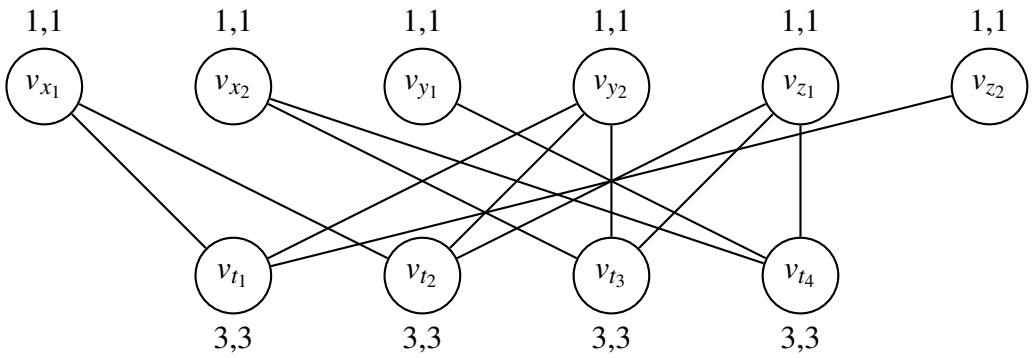

Fig. 4.3.1 An example of the reduction from 3DM-3 to MWBMMQ

We show that every feasible solution $M \subseteq T$ with value z to the given instance of 3DM-3 corresponds to a feasible solution to the instance of MWBMMQ as constructed above that has value $3z$, and vice versa:
We transform a solution $M \subseteq T$ to the instance of 3DM-3 into a solution to the instance of MWBMMQ by matching all edges that are incident to the nodes v_{t_j} for which $t_j \in M$.
Conversely, given a solution to MWBMMQ, we create a solution M to 3DM-3 by selecting those elements $t_j \in T$ for which v_{t_j} is matched.

We now show that the solutions that we obtain from the transformations described above are in fact feasible. We first consider the case of transforming a feasible solution to 3DM-3 into a solution to MWBMMQ: By construction, the constraints q and b are fulfilled regarding all nodes v_{t_j}, since v_{t_j} is either unmatched or matched by three edges. Assume that there is some node v_{x_i}, v_{y_i} or v_{z_i} that is matched more than once by M': W.l.o.g. we call this node v_{x_i}. Then, by construction, there are two elements in M that agree in the coordinate x_i, which contradicts feasibility of M.
Conversely, let a feasible solution to the instance of MWBMMQ be given and we create a solution to 3DM-3 as described above. We have to show that none of the elements $t_j \in M$ agree in any coordinate. Recall that we create exactly one node for each of the coordinates given by the instance of 3DM-3. Hence, if two elements in M agree in some coordinate, then the node corresponding to this coordinate is matched (at least) twice in the solution to MWBMMQ. Since the coordinates of the elements of t_j corresponds to one of the nodes v_{x_i}, v_{y_i} or v_{z_i} for which $b_v = 1$, this contradicts feasibility of the b-matching.
Hence, we have shown that the transformation described above works in both directions. We now consider the value of both solutions. Note that in every feasible solution to the instance of MWBMMQ the following holds:

- Every edge is incident to some node v_{x_i}, v_{y_i} and v_{z_i}. Hence, every edge can be used at most once in a feasible matching.
- Every edge is incident to some node v_{t_j}.
- Every node v_{t_j} is either unmatched or matched by three edges.
- None of the nodes v_{t_j} are adjacent to each other.

From these observations we conclude that the value of every solution to the given instance of MWBMMQ is exactly three times the number of matched nodes v_{t_j}. Furthermore, according to the transformation described above, a node v_{t_j} is matched if and only if $t_j \in M$. This yields the claim that the value of the solution to MWBMMQ is exactly three times the value of the corresponding solution to 3DM-3.
All in all, the correspondence between the solutions and their respective values of 3DM-3 and MWBMMQ has been established. Thus, MWBMMQ has the same hard gap as 3DM-3. □

The existence of the hard gap directly yields the following corollary regarding the approximability of MWBMMQ:

Corollary 4.3.2. *There is an $\epsilon > 0$ so that it is strongly NP-hard to approximate MWBMMQ within a factor larger than $1 - \epsilon$ (where ϵ is defined as above), even on bipartite graphs with $w_e = 1$ on all edges $e \in E$ and $q_v = b_v \leq 3$ for all $v \in V$. In particular, there is no PTAS for MWBMMQ.*

Remark 4.3.3. Alternatively, we could have shown the above result using a result from [2]. The authors show that MLQ (and thus WMLQ) is APX-complete using a reduction from the maximum independent set problem and, according to Remark 4.2.9, WMLQ is a special case of MWBMMQ.

In addition to the above result on general graphs, we now show that it is NP-hard to approximate MWBMMQ on series parallel graphs and on complete binary trees. In order to show NP-hardness of MWBMMQ on series parallel graphs, we use a reduction from PARTITION [16]:

Theorem 4.3.4. *It is NP-hard to approximate MWBMMQ on series parallel graphs and on bipartite graphs, even if $w_e = 1$ on all edges $e \in E$ and $q_v = b_v$ for all $v \in V$.*

Proof. Let an instance of PARTITION be given by n integers $s_i \in \mathbb{N}$ $(i = 1, \ldots, n)$ where $\sum_{i=1}^{n} s_i = B$.
We create a graph with node set $V = \{s, t, v_1, \ldots, v_n\}$, where each v_i corresponds to the respective integer s_i, and there are edges connecting each v_i with s and t. We set $w_e := 1$ for all edges $e \in E$, $q_{v_i} := b_{v_i} := s_i$ for $i = 1, \ldots, n$, $q_s := b_s := B/2$ and $q_t := b_t := B + 1$.
An example for $n = 5$ is given in Figure 4.3.2. It is obvious that we have created a series parallel graph. Since all edges that are used in a matching are incident to one of the nodes v_i, it follows that $\sum_{i=1}^{n} b_{v_i} = \sum_{i=1}^{n} s_i = B$ is an upper bound on the maximum value of every feasible matching. Furthermore, we get that the minimum quantity $q_t = B + 1$ cannot be achieved by any feasible matching. Thus, the edges that are incident to t are never used in any feasible matching and we can restrict our further analysis to the edges incident to s. In fact, these nodes have been added for technical reasons in order for the graph to be series parallel. This implies that every feasible b-matching has value either $B/2$ or 0.

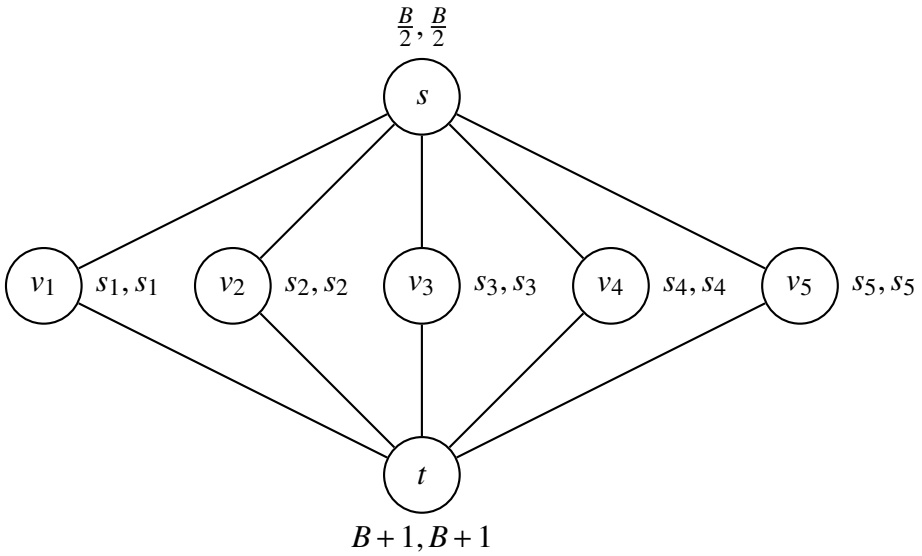

Fig. 4.3.2 An example of the reduction from PARTITION to MWBMMQ on a series parallel graph

Every b-matching with weight $B/2$ induces a solution to PARTITION if we group the integers of the given instance of PARTITION based on whether the corresponding node is matched. Vice versa, a solution to PARTITION induces a b-matching with weight $B/2$ if we match exactly those nodes that correspond to the integers that are in the first partition. Now assume that there is some α-approximation with $\alpha \geq 1$. If a solution to PARTITION exists, then there is also a b-matching with weight $B/2$ as we have just seen. In this case the α-approximation returns a matching with value at least $B/2\alpha > 0$. Due to the minimum quantity constraint on node s this implies that the matching actually has weight $B/2$ so that it induces a solution to PARTITION. If, on the other hand, there is no solution to PARTITION, then there is also no matching with weight $B/2$. As we have seen, the only feasible matching in this case is an empty matching. Hence, in this case the approximation returns an empty matching. Therefore, an α-approximation algorithm for MWBMMQ on series parallel graphs would solve PARTITION, which implies NP-hardness. □

As already stated above, the node t and the edges incident to it are never used in any feasible matching. Thus, the proof still works for the remaining tree if we omit the node t and the edges incident to it. Thus, we obtain the following result:

Corollary 4.3.5. *It is NP-hard to approximate MWBMMQ on trees, even if $w_e = 1$ on all edges $e \in E$ and $q_v = b_v$ for all $v \in V$.*

Note that while approximating MWBMMQ on bipartite graphs is NP-hard, WMLQ (which, by definition, implies bipartiteness of the graph) can be approximated in polynomial

time according to [2]. The approximation factor of the algorithm given in [2] depends on the instance of the problem. On the other hand, it is shown in [2] that deciding whether there is a solution with OPT> 0 is NP-hard for GMLQ on bipartite graphs. In particular, approximating GMLQ on bipartite graphs is NP-hard.

We now show that approximating MWBMMQ on complete binary trees is NP-hard (cf. Definition 1.4.12):

Theorem 4.3.6. *It is NP-hard to approximate MWBMMQ on complete binary trees, even if* $q_v = b_v$ *for all* $v \in V$.

Proof. As before, we use a reduction from PARTITION.
Let an instance of PARTITION be given by n integers s_i (for $i = 1,\dots,n$) where $\sum_{i=1}^{n} s_i = B$. We create an instance of MWBMMQ on a complete binary tree based on this instance of PARTITION. Set

$$h := 2\left\lceil \frac{\log n - 1}{2} \right\rceil + 1$$
$$n' := 2^h$$

Then by definition of h we have

$$\log n = 2\left(\frac{\log n - 1}{2}\right) + 1 \leq h < 2\left(\frac{\log n - 1}{2} + 1\right) + 1 = \log n + 2$$

and thus

$$n \leq n' < 4n$$

Expressed less technically, n' is the smallest odd power of 2 that is at least as large as n. We create a complete binary tree with n' leaf nodes. Note that in this case the height of the tree is h. The leaf nodes are called v_i ($i = 1,\dots,n$) and a_i ($i = 1,\dots,n' - n$). The nodes a_i are referred to as artificial nodes. The nodes v_i correspond to the integers s_i given by the instance of PARTITION. The root node is named r and the remaining nodes between the root and the leaves are referred to as t_j for some j. We now define the minimum quantities and capacities on the nodes as well as the weights on the edges.

If $w_{u,v}$ denotes the weight of the edge connecting the nodes u and v, we set

$$w_{u,v} := \begin{cases} 1 & \text{if } u = v_i \text{ or } v = v_i \text{ for some } i \in \{1, \dots, n\} \\ 0 & \text{else} \end{cases}$$

That is, the weights of the edges incident to the leaf nodes corresponding to the integers given by the instance of PARTITION are 1, the remaining edge weights are 0.

The minimum quantities q_v and the capacities b_v are defined as follows:

$$q_v := b_v := \begin{cases} s_i & \text{if } v = v_i \text{ for some } i \in \{1, \dots, n\} \\ \frac{B}{2} & \text{if } v = t_j \text{ for some } j \text{ and the depth of } t_j \text{ is even or } v = r \\ B & \text{if } v = t_j \text{ for some } j \text{ and the depth of } t_j \text{ is odd or } v = a_i \text{ for some } i \in \{1, \dots, n' - n\} \end{cases}$$

An example of the construction is given in Figure 4.3.3 and in Figure 4.3.4. Note that the height of the tree that we have constructed is always odd, which implies that the depth of the leaf nodes' parents is even and their minimum quantities and capacities are uniformly $B/2$. Thus, due to the minimum quantities and upper capacities of the artificial nodes a_i, which are uniformly B, none of the edges incident to them can ever be matched in any feasible matching. Their only purpose is to fill the complete binary tree structure.

We now show that there is a feasible solution to this instance of MWBMMQ with a strictly positive value if and only if there is a solution to the instance of PARTITION. In this case the solution to MWBMMQ induces a solution to PARTITION.

We can check in polynomial time whether any binary subtree T' induced by selecting some node and all its descendants fulfills $\sum_{v_i \in V(T')} b_{v_i} = B/2$, i.e. whether the sum of the capacities of its (non-artificial) leaf nodes v_i is $B/2$. If so, we have already found a feasible solution to PARTITION. Hence, in the following we can assume w.l.o.g. that no such subtree exists.

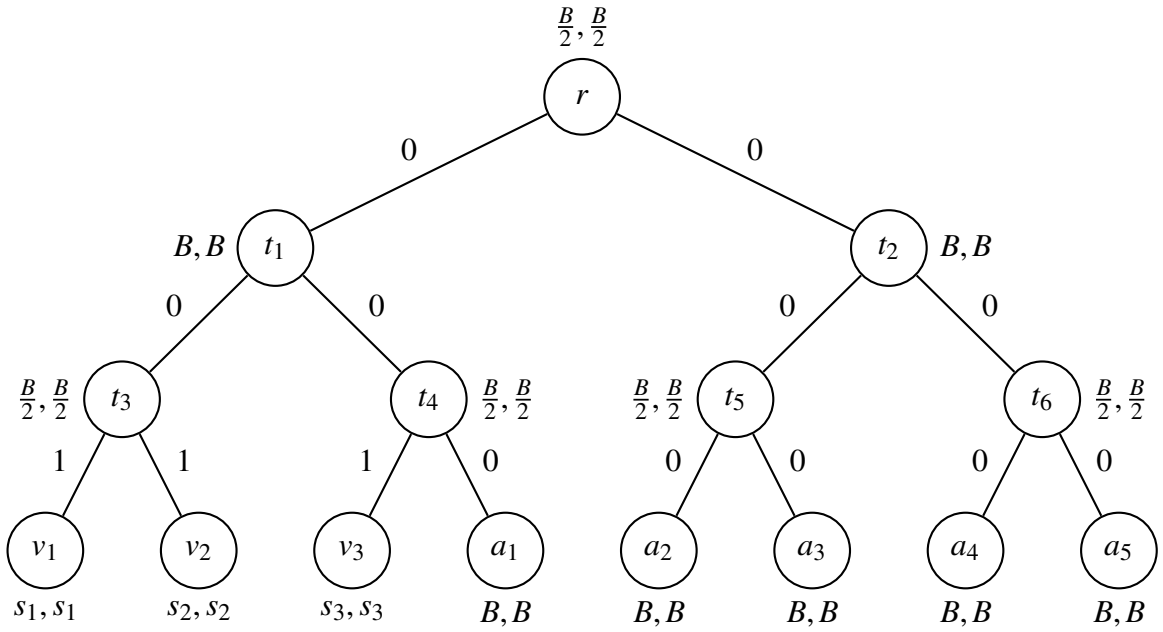

Fig. 4.3.3 An example of the construction of an instance of MWBMMQ on a complete binary tree with eight leaf nodes, five of which are artificial nodes.

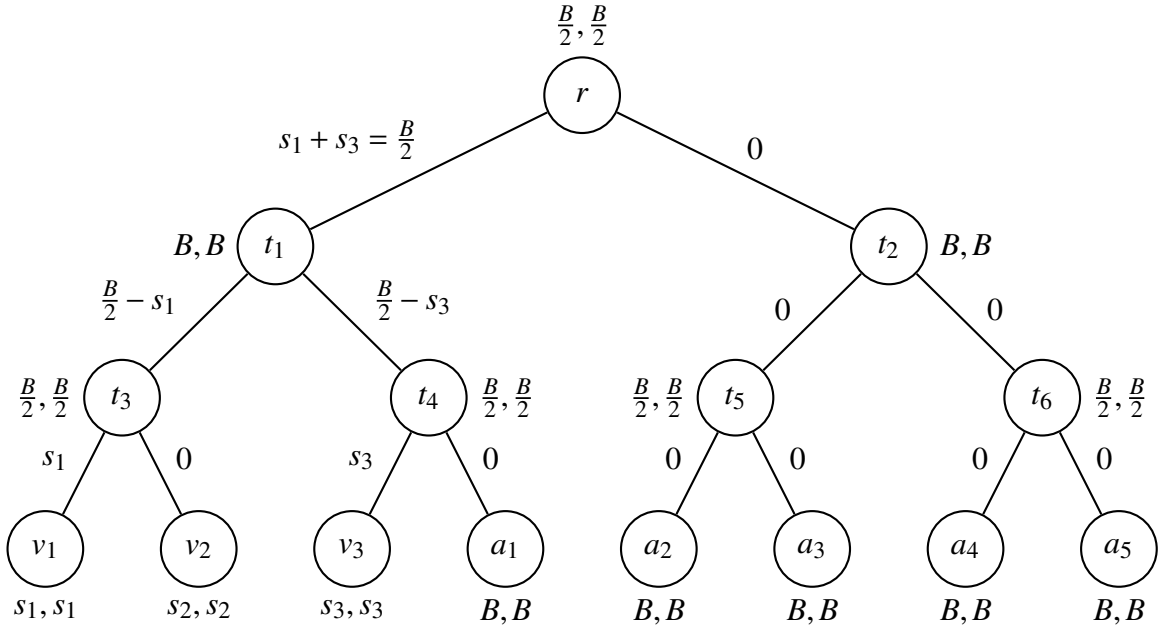

Fig. 4.3.4 The above example continued: Illustration of an optimal solution to MWBMMQ, assuming that $s_1 + s_3 = B/2 = s_2$ (unlike before, the edge labels describe the matching, not the edge weights)

We first show how, given a feasible solution to PARTITION, a feasible solution to the instance of MWBMMQ with strictly posive value can be created.
Let an index set $M \subseteq \{1,\dots,n\}$ with $\sum_{i\in M} s_i \leq B/2$ be given. Recall that the height h of the binary tree is always odd, so $h = 2k+1$ for some $k \in \mathbb{N}$. We show by induction on k how a matching can be constructed so that the following properties hold:

1. If $i \in M$ then $m(\delta(v_i)) = s_i$, else $m(\delta(v_i)) = 0$.
2. $m(\delta(r)) = \sum_{i\in M} s_i$
3. All minimum quantity constraints and capacity constraints (possibly) except for the minimum quantity constraint of the root node r are fulfilled.

For $k = 0$ the binary tree consists of the root node and two leaf nodes and by construction we have $s_1 + s_2 = B$. W.l.o.g. $1 \in M$. Setting $m(v_1, r) \coloneqq s_1$ and $m(v_2, r) \coloneqq 0$ yields the claim.
We now assume that the claim has already been shown for some k and we show the claim for $k+1$. We denote this binary tree of height $h = 2(k+1)+1$ by T. The tree is shown in Figure 4.3.5: The leaf nodes with thick borders represent complete binary trees of height $h = 2k+1$. For the sake of simplicity, only the root nodes r_j are displayed. We denote the respective subtree induced by r_j and its descendants by T_j and the set of its leaf nodes by L_j. Let $M_j \coloneqq \left\{i \in M \,|\, v_i \in L_j\right\}$ for all $j \in \{1,2,3,4\}$. Note that $\bigcup_{j=1}^{4} M_j = M$. By assumption, $\sum_{j=1}^{4} \sum_{i\in M_j} s_i = \sum_{i\in M} s_i \leq B/2$, which implies $\sum_{i\in M_j} s_i \leq B/2$ for all $j \in \{1,2,3,4\}$, i.e. we can apply the induction hypothesis to these subtrees of height $h = 2k+1$. Hence, there is a matching in each of the four subtrees T_j which fulfills the properties specified above. In particular, the minimum quantity constraints and the capacity constraints regarding all nodes in the subtrees T_j, possibly except for the root nodes r_j, are fulfilled. Since $\bigcup_{j=1}^{4} M_j = M$, the first property also holds for T. Furthermore, we have $m(\delta(r_j)) = \sum_{i\in M_j} s_i \leq B/2$. Setting $m_d \coloneqq B/2 - m(\delta(r_j))$ for $d \in \{1,2,3,4\}$ yields a matching where $m(\delta(r_j)) = B/2 = b_{r_j}$ for all $j \in \{1,2,3,4\}$, $m(\delta(t_1)) = B - m(\delta(r_1)) - m(\delta(r_2))$ and $m(\delta(t_2)) = B - m(\delta(r_3)) - m(\delta(r_4))$. Setting $m_5 \coloneqq m(\delta(r_1)) + m(\delta(r_2))$ and $m_6 \coloneqq m(\delta(r_3)) + m(\delta(r_4))$ yields a solution which is feasible for all nodes (possibly) except for r and where $m(\delta(r)) = \sum_{j=1}^{4} m(\delta(r_j)) = \sum_{j=1}^{4} \sum_{i\in M_j} s_i = \sum_{i\in M} s_i$, which yields the claim.
Now assume that a feasible solution to PARTITION exists: Choosing M in such a way that $\sum_{i\in M} s_i = B/2$ and applying the above procedure yields a matching for which $m(\delta(r)) = \sum_{i\in M} s_i = B/2$, so it is also feasible regarding the root node r. Furthermore, $m(\delta(v_i)) = s_i$ for all $i \in M$.

By construction, the edges incident to the leaf nodes have weight 1, which implies that the matching has weight $\sum_{i \in M} s_i = B/2 > 0$. Hence, if a solution to PARTITION exists, then there is a feasible solution to the instance of MWBMMQ with strictly positive value as claimed.

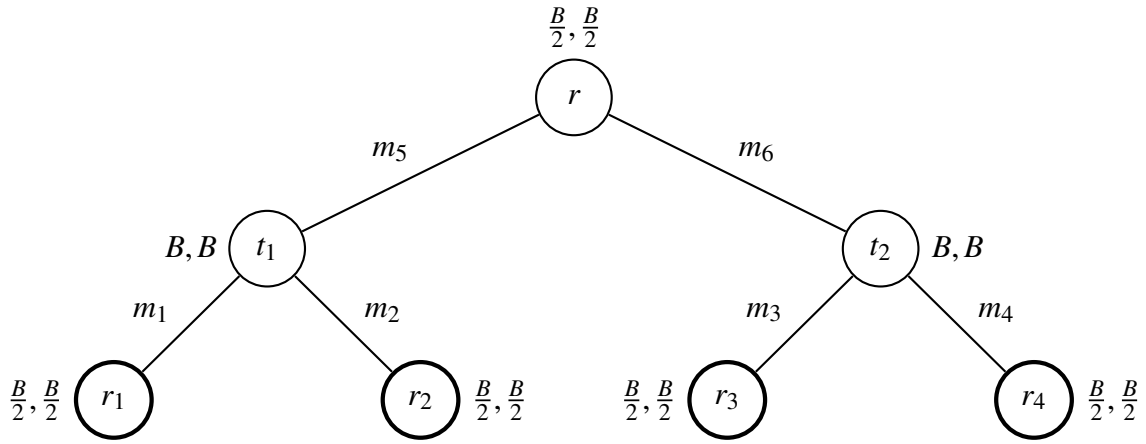

Fig. 4.3.5 A complete binary tree of height $h = 2(k+1)+1$. The leaf nodes with thick borders represent complete binary trees of height $h = 2k+1$.

Conversely, we now assume that there is a feasible matching m with strictly positive value. This implies that at least one of the leaf nodes is matched. We apply a simplification that will prove to be useful in the following argumentation:
We check (in polynomial time) whether there is any subtree T' that is induced by selecting some node $r' \neq r$ and all its descendants and that fulfills the following conditions:

- The induced subtree T' has odd height, which is equivalent to r' having even depth in the initial tree and which implies $q_{r'} = b_{r'} = B/2$.
- If we denote the restriction of the matching m to the edges in T' by m', then $m'(\delta(r')) = B/2$ holds.
- At least one leaf node of T' is matched by m'.

If there is any subtree that fulfills the above properties, we select a minimal subtree T' (i.e. one that does not contain any real subtree that fulfills these properties) and the corresponding matching m'. Note that m' is a feasible matching in T' with strictly positive value and that T' structurally differs from the initial tree only regarding its height. Thus, even if a subtree T' has been selected, we continue denoting the tree by T, its root node by r and the matching by m. If there is no subtree that fulfills these conditions, we continue with the initial tree.
In the following we assume w.l.o.g. that no such subtree exists (*).

Once again, let $h = 2k+1$ for some $k \in \mathbb{N}$ denote the height of T. We show by induction on k that if there is a matching m in the complete binary tree for which $m(\delta(r)) \leq B/2$ holds (i.e. the minimum quantity on r is relaxed) and that fulfills the constraints regarding all descendants of r, then

$$m(\delta(r)) = \begin{cases} 0 \text{ or } B/2 & \text{if } \sum_{i=1}^{n} m(\delta(v_i)) = 0 \\ \sum_{i=1}^{n} m(\delta(v_i)) & \text{if } \sum_{i=1}^{n} m(\delta(v_i)) > 0 \end{cases}$$

For $k = 0$, i.e. for the binary tree consisting of one root node and two leaf nodes, the claim follows immediately.

We now assume that the claim has been shown for some value k. For the induction step we consider Figure 4.3.5 again. As in the previous induction, we denote the subtrees with root nodes r_j and height $h = 2k+1$ by T_j. If we restrict the matching to the edges of one of these subtrees, then $m(\delta(r_j)) \leq B/2$ and the matching is still feasible for all descendants of r_j so that the induction hypothesis can be applied to the respective subtree. We denote the restriction of m to one of the subtrees by $m_{|T_j}$.

We assume that the first case $\sum_{i=1}^{n} m(\delta(v_i)) = 0$ holds for T, i.e. none of the leaf nodes is matched. Consequently, none of the leaf nodes of any of the subtrees T_j is matched and thus $m_{|T_j}(\delta(r_j)) \in \{0, B/2\}$ for all $j \in \{1,2,3,4\}$. Due to the constraints on the nodes r_j, we conclude that $m_d \in \{0, B/2\}$ for all $d \in \{1,2,3,4\}$. This implies that $m_d \in \{0, B/2, B\}$ for $d \in \{5,6\}$. By assumption we have $m(\delta(r)) \leq B/2$ and thus $m_5 + m_6 = m(\delta(r)) \in \{0, B/2\}$, which proves the first case of the claim.

We now assume that the second case, i.e. $\sum_{i=1}^{n} m(\delta(v_i)) > 0$, holds for T. This implies that at least one leaf node of T (and thus of T_j for at least one $j \in \{1,2,3,4\}$) is matched. We assume w.l.o.g. that there is at least one matched leaf node in the left branch starting at r (i.e. in the subtrees T_1 or T_2). Let L_j denote the set of leaf nodes of T_j. We prove the claim by showing that $m_5 = m(\delta(L_1 \cup L_2))$ and $m_6 = m(\delta(L_3 \cup L_4))$. We distinguish two cases that can occur in the left branch of the tree containing T_1 and T_2:

Case 1 Exactly one of the subtrees T_1 and T_2 contains at least one matched leaf node.
W.l.o.g. we assume that T_1 contains at least one leaf node that is matched. Then according to the induction hypothesis we have $m_{|T_1}(\delta(r_1)) = m(\delta(L_1)) > 0$. On the other hand, according to (*) we have $m_{|T_1}(\delta(r_1)) < B/2$. Thus, we get $0 < m_1 = B/2 - m_{|T_1}(\delta(r_1)) < B/2$. Due to the upper capacities on r and r_2 and the minimum quantity on t_1 we get that $m_2 > 0$ and $m_5 > 0$. By assumption T_2 does not contain any matched leaf nodes. This implies that $m_{|T_2}(\delta(r_2)) \in \{0, B/2\}$. So the only possible

value for m_2 is $B/2$. Due to the constraints on t_1 we get that $m_5 = B - m_1 - m_2 = B - (B/2 - m_{|T_1}(\delta(r_1))) - B/2 = m_{|T_1}(\delta(r_1)) = m(\delta(L_1)) = m(\delta(L_1 \cup L_2))$.

Case 2 Both subtrees T_1 and T_2 contain at least one matched leaf node.
By assumption and by (*) we get that $0 < m_1 = B/2 - m_{|T_1}(\delta(r_1)) < B/2$ and $0 < m_2 = B/2 - m_{|T_2}(\delta(r_2)) < B/2$. In order for the constraints on t_1 to be fulfilled, we get that $m_5 = B - (B/2 - m_{|T_1}(\delta(r_1))) - (B/2 - m_{|T_2}(\delta(r_2))) = m_{|T_1}(\delta(r_1)) + m_{|T_2}(\delta(r_2)) = m(\delta(L_1)) + m(\delta(L_2)) = m(\delta(L_1 \cup L_2))$.

For the right-hand side of the tree that contains the subtrees T_3 and T_4 these cases apply analogously so that we obtain $m_6 = m(\delta(L_3 \cup L_4))$ if we apply Case 1 and Case 2 to T_3 and T_4. Additionally, a third case is possible:

Case 3 None of the subtrees T_3 and T_4 contains any matched leaf node.
We have already shown that $m_5 > 0$. Taking into account that $m(\delta(r)) \leq B/2$ by assumption, this implies that $m_6 < B/2$. Considering the capacity and minimum quantity constraints on t_2, r_3 and r_4 as well as the fact that by assumption $m_{|T_i}(\delta(r_i)) \in \{0, B/2\}$ for $i \in \{3,4\}$, we get that $m_6 = 0 = m(\delta(L_3 \cup L_4))$.

Hence, we can conclude that $m(\delta(r)) = m_5 + m_6 = m(\delta(L_1 \cup L_2)) + m(\delta(L_3 \cup L_4)) = \sum_{i=1}^{n} m(\delta(v_i))$, which shows the second case of the claim and ends our induction.
By assumption there is a feasible matching m with a strictly positive value, hence we get that $m(\delta(r)) = \sum_{i=1}^{n} m(\delta(v_i)) > 0$. Note that this matching also fulfills the minimum quantity constraint on r. Therefore, we get that $\sum_{i=1}^{n} m(\delta(v_i)) = m(\delta(r)) = B/2$. By construction, in order for the matching to be feasible, every leaf node v_i is either unmatched or matched by $q_{v_i} = b_{v_i} = s_i$. Hence, the matched leaf nodes induce a solution to PARTITION.
We have shown that every solution with strictly positive value induces a solution to PARTITION and vice versa. Now assume that there is an approximation algorithm for MWBMMQ on complete binary trees. If the instance of PARTITION that we consider is solvable, then there is a solution to the instance of MWBMMQ with strictly positive value. So the approximation algorithm also returns a solution with strictly positive value, which induces a solution to PARTITION, as we have just seen. Conversely, if there is no solution to the instance of PARTITION, then the only feasible solution to the instance of MWBMMQ is an empty matching, which is returned by the approximation algorithm. Therefore, an approximation algorithm for MWBMMQ on complete binary trees would solve PARTITION, which completes the proof. □

Note that none of the above theorems 4.3.1, 4.3.4 and 4.3.6 requires the integrality of the matching x. So we would obtain the same results if we changed Definition 4.2.5 by requiring only $m \in \mathbb{Q}^{|E|}$.

Remark 4.3.7. Note that all hardness results concerning MWBMMQ also apply to MWECB-MMQ as it is a generalization of MWBMMQ.

Having shown NP-hardness of MWBMMQ (and of MWECBMMQ) and several variants of the problem, we now focus on solving or approximating variants of MWBMMQ (and of MWECBMMQ) in (pseudo-)polynomial time.
Be begin by showing how MWBMMQ can be solved in polynomial time in a straightforward way if the number of nodes is bounded:

Theorem 4.3.8. *For graphs where $V(G) \leq k$ for $k \in \mathbb{N}$ fixed, MWBMMQ can be solved in polynomial time.*

Proof. We select a subset of the nodes $V' \subseteq V$ and create an instance of the maximum-weight $[a',a'']$-matching problem on $G(V')$ by interpreting each minimum quantity as a fixed lower bound (as described in Definition 4.2.3). According to 4.2.4, this instance can be solved in polynomial time. By trying all possible $V' \subseteq V$ and choosing a solution that maximizes the weight, we find an optimal solution to the initial problem. Since the number of possible subsets of V is bounded by 2^k and k is a constant, the overall runtime of this procedure is also polynomial. □

We now show that MWBMMQ can be solved in polynomial time on paths and cycles.

Theorem 4.3.9. *MWBMMQ can be solved in polynomial time on paths.*

Proof. Let an instance of MWBMMQ on a path consisting of n nodes be given. We provide a dynamic program that solves the problem in polynomial time.
Let G denote the graph and let $V(i,j) \coloneqq \{i, i+1, \dots, j\} \subseteq V$ for $i \leq j$ and $i, j \in \{1, \dots, n\}$. Analogously, we denote the subpath induced by $V(i,j)$ by $G(i,j) \coloneqq G[V(i,j)]$ and its edge set by $E(i,j) \coloneqq E(G(i,j))$.
In order to construct the dynamic program, we require two values that we compute and store during the execution of the dynamic program.
The first value $M(i,j)$ is the value of a maximum-weight $[q,b]$-matching in $G(i,j)$ in the sense of Definition 4.2.3. According to Theorem 4.2.4, this value can be computed in polynomial time. We compute $M(i,j)$ for all subpaths of G, which implies that $n(n-1)/2$ computations of $M(i,j)$ are necessary: There is one path of length $n-1$, two paths of length $n-2$ and so on. Hence, the number of tuples (i,j) for which we have to compute $M(i,j)$ is polynomially

bounded in the size of the input instance.
The second value we require is the optimal solution to MWBMMQ on $G(1, j)$, which we denote by $M(j)$. Set $M(0) := 0$. Note that according to this definition, $M(n)$ is the optimal value we are looking for.
The dynamic program works as follows: We first compute all $M(i, j)$ as defined above. For $j \in \{1, \dots n\}$ we successively compute

$$M(j) := \max \left\{ M(1, j), \max_{2 \le i < j} (M(i, j) + M(i-2)), \max_{1 \le i < j} M(i) \right\}.$$

In fact, this computes the optimal solution to MWBMMQ on $G(1, j)$ as defined above:
First of all note that once we compute $M(j)$ for some j, all values used in the above equation are already known. An optimal solution on $G(1, j)$ can be structured in one of the following three ways: Firstly, it is possible that all nodes $\{1, \dots, j\}$ are matched. In this case, all the minimum quantity constraints must be satisfies and thus the matching corresponds to $M(1, j)$. Secondly, node j might be matched but there is at least one node in $G(1, j)$ that is not matched. Note that in this case, node $j-1$ must be matched as well. Let i be the node with the smallest index for which all nodes in $G(i, j)$ are matched. We can neglect the case $i = 1$ because this implies that *all* nodes are matched and we have already considered this case. Then node $i - 1 \ge 1$ is unmatched and thus an optimal solution on $G(i, j)$ is the sum of two partial solutions that can be optimized independently: The value of an optimal matching on $G(i, j)$ that matches all nodes is exactly $M(i, j)$. The optimal solution on the graph $G(1, i-2)$ is $M(i-2)$. Note that $G(1, i-2) = G(1, 0)$ for $i = 2$ is an empty graph and we have defined $M(0) = 0$. Since we do not know the node with the smallest index for which all nodes in $G(i, j)$ are matched, we try all possible values i and choose the maximum among the corresponding solutions. This case is reflected by the second term. Thirdly, the node j might not be matched in an optimal solution on $G(1, j)$. So the optimal solution on $G(1, j)$ is the maximum optimal solution on the subpaths $G(1, i)$ where $i < j$ that have already been computed in previous steps and where $M(i)$ is the corresponding objective value.
All in all, we have shown inductively, that the value of an optimal solution is in fact $M(n)$.
We have already pointed out that we can compute all $M(i, j)$ in polynomial time. Furthermore, $M(j)$ for a specific value of j can be computed in polynomial time and $j \in \{1, \dots, n\}$. So all in all, the dynamic program runs in polynomial time. □

From the above algorithm we can easily derive a polynomial algorithm for solving MWBMMQ on cycles:

Corollary 4.3.10. *MWBMMQ can be solved in polynomial time on cycles.*

Proof. Let an instance of MWBMMQ on a cycle be given. Trivially, in an optimal solution either all nodes are matched or there is at least one node that is unmatched. Assuming that all nodes are matched, we can find an optimal solution in polynomial time according to Theorem 4.2.4. If at least one node is unmatched, then the value of an optimal solution on the cycle is the same as on the path that we obtain by removing the respective node and the two edges incident to it. There are n such paths for which we have to compute an optimal solution. Choosing the optimum among these $n+1$ solutions yields an optimal solution to MWBMMQ on the given cycle. □

Applying a result regarding GWMLQ from [2], we obtain the following result:

Theorem 4.3.11. *MWBMMQ can be solved in polynomial time if $b_v \leq 2$ for all $v \in V$.*

Proof. According to [2], GWMLQ can be solved in polynomial time if $b_v \leq 2$ for all $v \in V$. Given an instance of MWBMMQ for which $b_v \leq 2$ for all $v \in V$, an edge can be used at most twice in each feasible solution. We transform the instance of MWBMMQ into an equivalent instance of GWMLQ by creating an additional edge $e' = (v, w)$ for each edge $e = (v, w)$ of the original graph. Now every feasible solution to the instance of MWBMMQ can be mapped to a feasible solution to the instance of GWMLQ that has the same objective value and vice versa. Thus, applying the algorithm given in [2] induces an optimal solution to the instance of MWBMMQ. □

We now show for MWECBMMQ that an optimal solution can be found in pseudo-polynomial time if we restrict our analysis to graphs of bounded treewidth:

Theorem 4.3.12. *Let a fixed integer $t \geq 1$ be given. MWECBMMQ can be solved in pseudo-polynomial time on graphs for which the treewidth is at most t.*

Proof. We show that an optimal solution can be computed using the following dynamic program. As we have already seen, with the treewidth being bounded from above by a constant, a nice tree-decomposition can be computed in linear time (Theorem 1.4.9).
For calculating and storing the intermediate solutions of the dynamic program, we require the following values for each node i of the tree-decomposition:
By $val_i^{(1)}(x)$ where $x \in \mathbb{N}^{|X_i|}$ and $x_{v_k} \in [0, b_v]$ for $v_k \in X_i$ we denote the value of a maximum-weight b-matching m in G_i that fulfills:

- $m(\delta(v_k)) = x_{v_k}$ for all $v_k \in X_i$
- $m(\delta(v_k)) \in \{0\} \cup [q_{v_k}, b_{v_k}]$ for all $v_k \in V_i \setminus X_i$
- $m_e \in \{0\} \cup [q_e, b_e]$ for all $e \in E_i$

Hence, the vector x describes $m(\delta(v_k))$ for the nodes $v_k \in X_i$ in the respective intermediate solution m. The minimum quantity constraints for these nodes may be violated at this point in time. For the nodes $v_k \in V_i \setminus X_i$ we require the minimum quantity and capacity constraints to be fulfilled by the intermediate solution. The constraints on the edges always have to be fulfilled.
We define $val_i^{(2)}(x)$ just as $val_i^{(1)}(x)$ except for the additional constraint that the matching m may not use any edge $e = (v, w)$ where $v, w \in X_i$.
The dynamic program computes these values along the tree-decomposition, starting at the leaf nodes and moving towards the root node. The computation of a value at some node i of the tree-decomposition requires that the values have been computed for all child nodes of the respective node.
Once the dynamic program considers some node i of the tree-decomposition, all the edges that are incident to the nodes in $V_i \setminus X_i$ in the original graph G have already been considered. This explains why we require the minimum quantity constraints of these nodes to be fulfilled when computing $val_i^{(1)}(x)$ and $val_i^{(2)}(x)$. For the nodes in X_i further edges might be considered in subsequent steps, so that the minimum quantity constraints might be fulfilled once the dynamic program terminates even if they are violated in an intermediate solution. For each node i of the tree-decomposition all values $val_i^{(1)}(x)$ and $val_i^{(2)}(x)$, where $x \in \mathbb{N}^{|X_i|}$ and $x_v \leq b_v$ for all $v \in X_i$, must be computed. We describe the computation of $val_i^{(1)}(x)$ and $val_i^{(2)}(x)$ for each node type in the tree-decomposition. For the complexity analysis we use the notation $n := |V(G)|$ and $b_{\max} := \max_{v \in V(G)} b_v$. Note that $|X_i| \leq t + 1$ for all nodes i of the tree-decomposition.

- Leaf node
  For the leaf nodes of the nice tree-decomposition we have that $|X_i|$=1. Let $v_k \in X_i$. Obviously, G_i does not contain any edges so that $x_{v_k} = x = 0$ is the only value of x for which a feasible intermediate solution m with $m(\delta(v_k)) = x_{v_k} = 0$, i.e. an empty matching, exists. To indicate that there is no feasible matching for some vector x, we use the value $-\infty$.

$$val_i^{(1)}(x) := val_i^{(2)}(x) := \begin{cases} 0 & \text{if } x = 0 \\ -\infty & \text{else} \end{cases}$$

The computation of $val_i^{(1)}$ and $val_i^{(2)}$ for a fixed value x can be carried out in constant time. Considering all possible values of x yields the computational complexity $O(b_{\max})$.

- Insert node ($X_i = X_j \cup \{v\}$ for some i, j, v)
  Set $f := \left|X_j\right| = |X_i| - 1 \leq t$ and $g := \left|\left\{v_k \in X_j \,|\, (v, v_k) \in E(G_i)\right\}\right| \leq f \leq t$. Let $\left\{v_1, \ldots, v_g\right\}$ denote the nodes in G_i that are incident to v and $\left\{v_{g+1}, \ldots, v_f\right\}$ those that are not.

  For an insert node i, $val_i^{(1)}\begin{pmatrix} x \\ x_v \end{pmatrix}$ can be decomposed into a matching on the edges connecting the nodes in X_j with v and a matching in G_j. For the edges in G_i that are incident to v we try all feasible matchings $m \in \mathbb{N}^g$ where $m(\delta(v)) = x_v$. For $1 \leq k \leq g$ we denote the matching on the edge $e_k = (v, v_k)$ by m_k and the respective edge weight by w_k. In addition, we use the notation $I_k := \{0\} \cup [q_{e_k}, b_{e_k}]$.
  Maximizing the sum of both partial matchings yields:

$$val_i^{(1)}\begin{pmatrix} x \\ x_v \end{pmatrix} = val_i^{(1)}\begin{pmatrix} x_1 \\ \vdots \\ x_g \\ x_{g+1} \\ \vdots \\ x_f \\ x_v \end{pmatrix} := \max_{\substack{m \in \mathbb{N}^g: \\ m_k \in I_k \text{ for all } 1 \leq k \leq g \wedge \\ m_k \leq x_k \text{ for all } 1 \leq k \leq g \wedge \\ \sum_{k=1}^{g} m_k = x_v}} \left(\sum_{k=1}^{g} m_k w_k + val_j^{(1)}\begin{pmatrix} x_1 - m_1 \\ \vdots \\ x_g - m_g \\ x_{g+1} \\ \vdots \\ x_f \end{pmatrix} \right)$$

  There are at most $(b_{\max} + 1)^{f+1}$ feasible values for $\begin{pmatrix} x \\ x_v \end{pmatrix} \in \mathbb{N}^{f+1}$. For a fixed value $\begin{pmatrix} x \\ x_v \end{pmatrix}$, we consider a maximum over at most $(b_{\max} + 1)^g$ matchings m. If both $\begin{pmatrix} x \\ x_v \end{pmatrix}$ and m are fixed, the above term can be evaluated in constant time, since it is the sum of $g + 1$ terms, $g \leq t$ and t is fixed by assumption. Taking into account that $g \leq f \leq t$, we obtain the computational complexity $O(b_{\max}^{2t+1})$.

  By definition, for determining $val_i^{(2)}$ we only have to consider matchings in G_i in which none of the edges connecting two nodes in X_i is used. Hence, the node v cannot be matched, i.e. $m(\delta(v)) = 0$, since in G_i it is only adjacent to nodes in X_i.

So we can derive $val_i^{(2)}$ from $val_j^{(2)}$ in the following way:

$$val_i^{(2)}\begin{pmatrix} x \\ x_v \end{pmatrix} := \begin{cases} val_j^{(2)}(x) & \text{if } x_v = 0 \\ -\infty & \text{else} \end{cases}$$

This computation can be performed in constant time for a fixed value $\begin{pmatrix} x \\ x_v \end{pmatrix} \in \mathbb{N}^{f+1}$. Considering all possible values of $\begin{pmatrix} x \\ x_v \end{pmatrix}$ yields the computational complexity $O(b_{\max}^{t+1})$.

- Forget node ($X_i = X_j \setminus \{v\}$ for some i, j, v)
  Set $f := |X_i| = \left|X_j\right| - 1 \leq t$ and $g := \left|\left\{v_k \in X_i \,|\, (v, v_k) \in E(G_j)\right\}\right| \leq f \leq t$.
  In one of the previous iterations we have computed $val_j^{(1)}$ for all relevant values. Note that $G_i = G_j$. The set of matchings that we consider for computing $val_i^{(1)}$ is a subset of those matchings we have considered for $val_j^{(1)}$. Since $v \notin X_i$, we require $m(\delta(v)) \in \{0\} \cup [q_v, b_v]$. Thus, we can compute $val_i^{(1)}$ using the values for $val_j^{(1)}$ in the following way:
  $$val_i^{(1)}(x) := \max_{x_v \in \{0\} \cup [q_v, b_v]} val_j^{(1)}\begin{pmatrix} x \\ x_v \end{pmatrix}$$
  For a fixed value $x \in \mathbb{N}^f$, we consider the maximum over at most $b_{\max} + 1$ values. There are at most $(b_{\max} + 1)^t$ values for x to be considered, so the overall complexity of computing $val_i^{(1)}$ for a fixed i is $O(b_{\max}^{t+1})$.
  The computation of the values of $val_j^{(2)}$ considered only matchings that do not use any edges that connect two nodes in X_j. In particular, the matchings did not use any edges $e = (v, w)$ where $w \in X_j$, since $v \in X_j$. On the other hand, in order to compute $val_i^{(2)}$, edges $e = (v, w)$ where $w \in X_i = X_j \setminus \{v\}$ may be used by the intermediate solution, since $v \notin X_i$. Thus, by decomposing a matching in G_i into a matching on the edges connecting node v with one of the nodes in X_i and a matching that does not use any edges that connect two nodes in X_j we can reuse the values of $val_j^{(2)}$ for the computation of $val_i^{(2)}$. As for the insert node, let $\left\{v_1, \ldots, v_g\right\}$ denote the nodes in G_i that are incident to v and $\left\{v_{g+1}, \ldots, v_f\right\}$ those that are not. Again, for some matching m, we denote the matching

on $e_k = (v, v_k)$ by m_k and the respective weight by w_k. Let I_k be defined as before.

$$val_i^{(2)}\begin{pmatrix} x_1 \\ \vdots \\ x_g \\ x_{g+1} \\ \vdots \\ x_f \end{pmatrix} := \max_{\substack{m \in \mathbb{N}^g \wedge x_v \in \{0\} \cup [q_v, b_v]: \\ m_k \in I_k \text{ for all } 1 \le k \le g \wedge \\ m_k \le x_k \text{ for all } 1 \le k \le g \wedge \\ \sum_{k=1}^{g} m_k \le x_v}} \left(\sum_{k=1}^{g} m_k w_k + val_j^{(2)}\begin{pmatrix} x_1 - m_1 \\ \vdots \\ x_g - m_g \\ x_{g+1} \\ \vdots \\ x_f \\ x_v - \sum_{k=1}^{g} m_k \end{pmatrix} \right)$$

The complexity of this computation can be determined similarly as the complexity of determining $val_i^{(1)}$ of an insert node: Since $g \le f \le t$, the maximum is created over at most $(b_{\max} + 1)^t$ matchings m and at most $b_{\max} + 1$ values x_v. So for fixed x, $val_i^{(1)}(x)$ can be computed in $O(b_{\max}^{t+1})$. There are at most $(b_{\max} + 1)^t$ values x for which $val_i^{(2)}$ has to be determined, so overall we obtain $O(b_{\max}^{2t+1})$.

- Join node ($X_i = X_r = X_s$ for some i, r, s, where X_r and X_s are the child nodes of X_i)
  Set $f := |X_i| \le t + 1$ and $g := |E(G[X_i])|$.
  In order to compute the values of $val_i^{(1)}$, we note that every matching in G_i can be decomposed into three edge-disjoint matchings: Two matchings in G_r and G_s respectively, where none of the edges in $E(G[X_i])$ is used and where the constraints on all nodes that are not in X_i are satisfied and a matching m in $G[X_i]$. For an edge $e_k \in E(G[X_i])$ we denote the matching and the weight on the edge by m_k and w_k respectively. Again, we set $I_k := \{0\} \cup [q_{e_k}, b_{e_k}]$.

$$val_i^{(1)}(x) := \max_{\substack{m \in \mathbb{N}^g \wedge y, z \in \mathbb{N}^f: \\ m_k \in I_k \text{ for all } 1 \le k \le g \wedge \\ m(\delta(v)) + y_v + z_v = x_v \text{ for all } v \in X_i}} \left(\sum_{k=1}^{g} m_k w_k + val_r^{(2)}(y) + val_s^{(2)}(z) \right)$$

  Note that G is assumed to be simple. Hence, the number of edges g is bounded from above by $f(f-1)/2 \le t(t+1)/2$. This implies that the number of matchings m to be considered when determining the above maximum is bounded from above by $(b_{\max} + 1)^{t(t+1)/2}$. For a given matching m, there are at most $(b_{\max} + 1)^{t+1}$ combinations of y and z to be considered. So all in all, the maximum is created over at most $(b_{\max} + 1)^{(t+2)(t+1)/2}$ values. Considering up to $(b_{\max} + 1)^{t+1}$ values for x, we obtain an overall complexity of $O\left(b_{\max}^{(t+4)(t+1)/2}\right)$.
  For the computation of $val_i^{(2)}$ recall that none of the edges in X_i may be used in the

respective matchings. Since $X_i = X_r = X_s$, this also holds for $val_r^{(2)}$ and $val_s^{(2)}$. By definition, $V_r \cap V_s = X_i$. This enables us to reuse the values $val_r^{(2)}$ and $val_s^{(2)}$:

$$val_i^{(2)}(x) := \max_{\substack{y \in \mathbb{N}^f: \\ y_v \leq x_v \text{ for all } v \in X_i}} \left(val_r^{(2)}(y) + val_s^{(2)}(x-y) \right)$$

For x fixed, the maximum is created over at most $(b_{\max}+1)^{t+1}$ values of y and there are at most $(b_{\max}+1)^{t+1}$ possible values for x to be considered, so we obtain the computational complexity $O\left(b_{\max}^{2t+2}\right)$.

Once all these values have been computed for all nodes of the tree-decomposition, an optimal solution can be determined in the following way, where r is the root node of the binary tree and $f := |X_r| \leq t+1$:

$$\text{OPT} := \max_{\substack{x \in \mathbb{N}^f: \\ x_v \in \{0\} \cup [q_v, b_v] \text{ for all } v \in X_r}} \max \left\{ val_r^{(1)}(x), val_r^{(2)}(x) \right\}$$

This value can be computed in $O\left(b_{\max}^{t+1}\right)$. We summarize our complexity considerations as follows:

- A nice tree-decomposition of the input graph can be computed in linear time and its number of nodes is linearly bounded in the number of nodes of the input graph (Theorem 1.4.9).
- As the dynamic program moves through the tree-decomposition, all values $val_i^{(1)}$ and $val_i^{(2)}$ for each node i of the tree-decomposition can be computed in pseudo-polynomial time (as shown above).
- Once we have reached the root node, we can determine an optimal solution in pseudo-polynomial time.

All in all, we can conclude that the complexity of the above dynamic program is in fact pseudo-polynomial. The correctness of the algorithm follows from the reasoning provided for each of the cases. □

Remark 4.3.13. A similar dynamic program for GWMLQ is given in [2]. The dynamic program solves GWMLQ in time $O\left(T + b_{\max}^{3(t+1)}|E|\right)$, where T denotes the time required for computing a tree-decomposition of width t. The authors assume that there are no parallel edges, so w.l.o.g. $b_{\max}$ is bounded from above by $|V|$. Given that the tree-decomposition can be computed in linear time, they obtain a polynomial-time dynamic program for GWMLQ if

the treewidth of the underlying graph is bounded. Note that the dynamic program is even linear if we restrict it to those instances of GWMLQ for which $b_{\max}$ is bounded by a constant. On the other hand, the dynamic program for MWBMMQ given in the previous proof cannot be polynomial unless P=NP (Corollary 4.3.5).

Remark 4.3.14. Note that an instance of RSAP as defined in Chapter 2 can easily be transformed into an instance of WMLQ (Definition 4.2.7) and vice versa. The same holds for RPMP and MLQ (Definition 4.2.7). Thus, all results regarding WMLQ also apply to RSAP and all results regarding MLQ apply to RPMP (and vice versa).

In particular, this allows us to apply the results from Remark 4.3.13 and we obtain the following corollary:

Corollary 4.3.15. *RSAP (and thus RPMP) can be solved in polynomial time if the treewidth of the corresponding feasibility graph is bounded by a constant. If, in addition, the maximum capacity* $\max_{v \in V} b_v$ *is bounded by a constant, RSAP (and thus RPMP) can be solved in linear time.*

From the proof of theorem 4.3.12 we also get the following corollary:

Corollary 4.3.16. *Let fixed integers* $b, t \geq 1$ *be given. MWECBMMQ can be solved in linear time on graphs for which the treewidth is at most* t *and for which the maximum capacity* $\max_{v \in V} b_v$ *is bounded by* b*.*

Proof. Given the additional assumption that $\max_{v \in V} b_v$ is bounded, we consider the computational complexity of the algorithm given in the proof of theorem 4.3.12 again:

- A nice tree-decomposition of the input graph can be computed in linear time (Theorem 1.4.9).
- For each of the nodes of the tree-decomposition we compute $val_i^{(1)}$ and $val_i^{(2)}$. The number of nodes of the tree-decomposition is linearly bounded in the number of nodes of the input graph (Theorem 1.4.9). The computations at each of the nodes requires $O(1)$ steps.
- Once we have reached the root node, an optimal value can be computed in time $O(1)$.

□

Note that trees have treewidth one and that series parallel graphs have treewidth at most two (Lemma 1.4.11). Combining the existence of a pseudo-polynomial dynamic program with NP-hardness of the problem (Theorem 4.3.4 and Corollary 4.3.5) yields *weak* NP-hardness of the problem:

Corollary 4.3.17. *MWBMMQ is weakly NP-hard on trees and on series parallel graphs.*

We continue examining the solvability of MWBMMQ on graphs with a special structure and consider series parallel graphs. Recall that we have already shown NP-hardness on this type of graphs (Theorem 4.3.4). We now show that there is a polynomial-time bicriteria (α, β)-approximation for an arbitrary $\epsilon > 0$ even for MWECBMMQ (Definition 4.2.6). In this case, β refers to the minimum quantity and capacity constraints defined on the nodes *and* on the edges:

Theorem 4.3.18. *For $\epsilon > 0$ fixed, there is a polynomial-time $(1, 1+\epsilon)$-approximation for MWECBMMQ on series parallel graphs.*

Proof. The following proof gives a dynamic program that finds an approximate solution as described in the claim. Note that by definition of MWECBMMQ, negative edge weights are allowed. However, this does not create any ambiguity regarding our definition of the approximation ratio: Since an empty matching is a feasible solution with value 0, the optimal objective value is always non-negative.
By definition, every series parallel graph G can be constructed by a sequence of series and parallel compositions of series parallel graphs. Note that there might be multiple possible ways of constructing the given series parallel graph. In the following we assume that an arbitrary but fixed way of constructing the given graph is chosen. Our proof of the quality of the algorithm does not rely on this choice. In order to achieve a polynomial complexity of the algorithm, we only consider a subset of all feasible matchings during each construction step of the series parallel graph. The subset of matchings to be considered by the algorithm is characterized by the sets $J_{G',s}$ and $J_{G',t}$, where G' denotes the subgraph of G constructed in the current iteration and s and t denote its terminals. For each tuple $(a,b) \in J_{G',s} \times J_{G',t}$ the algorithm determines whether there is a matching m in G' that fulfills certain conditions and for which $m(\delta(s)) = a$ and $m(\delta(t)) = b$. If so, one such matching is chosen to be stored. Each of the matchings stored for a certain subgraph can be unambiguously identified by a corresponding tuple (a,b). In addition, there is a value function $W_{G'} : J_{G',s} \times J_{G',t} \to \mathbb{Q}$ that corresponds to the value of the partial solution, i.e. the weight of the matching. The crucial step of the algorithm is the cleansing step that is carried out after each parallel composition. The set of possible solutions that is stored for the next iteration is narrowed down to certain

representatives. Due to this step, the complexity of the algorithm remains polynomial.
Let an instance of MWECBMMQ be given on a graph $G = (V, E)$.
Set

- $b_{\max} := \max_{i \in V} b_i$
- $\sigma := \sqrt[n]{1+\epsilon} - 1$ where $n = |V|$
- $k := \lceil \log_{1+\sigma} b_{\max} \rceil = \lceil n \log_{1+\epsilon} b_{\max} \rceil$

This implies $k \in O(n \log \max_{i \in V} b_i)$, so it is polynomial in the size of the given instance of MWECBMMQ. In addition, we define the following intervals and values:

- $I_0 := \{0\}$
- $I_i := \left[(1+\sigma)^{i-1}, (1+\sigma)^i\right) \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{i-1} \right\rceil, \left\lceil (1+\sigma)^i \right\rceil - 1\right]$ for all $1 \leq i \leq k-1$
- $I_k := \left[(1+\sigma)^{k-1}, (1+\sigma)^k\right] \cap \mathbb{Z} = \left[\left\lceil (1+\sigma)^{k-1} \right\rceil, \left\lfloor (1+\sigma)^k \right\rfloor\right]$
- $p_0 := 0$
- $p_i := \left\lceil (1+\sigma)^{i-1} \right\rceil$ for all $1 \leq i \leq k$
- $p_{k+1} := \left\lfloor (1+\sigma)^k \right\rfloor$

This allows us to denote the intervals I_i using the values p_j:

- $I_0 = \{p_0\} = \{0\}$
- $I_i = [p_i, p_{i+1} - 1]$ for all $1 \leq i \leq k-1$
- $I_k = [p_k, p_{k+1}]$

We make the following observations regarding the above definitions:

- The intervals I_i might be empty for some $2 \leq i \leq k$.
- We have $\{0, 1, \ldots, b_{\max}\} \subseteq \bigcup_{i=0}^{k} I_i$.
- For all $1 \leq i \leq k$ for which $I_i \neq \emptyset$ we have $\frac{\max I_i}{\min I_i} \leq 1 + \sigma$.

We now describe how the algorithm works for each of the three possible situations which are the basic one-edge graph K_2, the series composition and the parallel composition. We assume that graph G' with terminal nodes s and t is the graph to be constructed in the current iteration. We use the same index for each series parallel graph and its respective source and sink, i.e. the terminals of graph G_i are denoted by s_i and t_i.

- $G' = K_2$
  We set $J_{G',s} \coloneqq J_{G',t} \coloneqq \{p_i \mid i = 0, \ldots, k+1\}$ so that $\left|J_{G',s}\right| = \left|J_{G',t}\right| \leq k+2 < (k+1)^2$. The first inequality follows from the fact that some of the p_i might coincide and the second inequality holds since $k \geq 1$. We denote the (only) edge of the graph by $e = (s,t)$. We define the value function as follows:

$$W_{G'}(a,b) \coloneqq \begin{cases} a \cdot w_e & \text{if } a = b \wedge a \leq \min\{b_s, b_t\} \cdot (1+\sigma) \\ & \wedge\, a \in \{0\} \cup \left[\frac{q_e}{1+\sigma}, (1+\sigma) b_e\right] \\ -\infty & \text{else} \end{cases}$$

  There are at most $(k+2)^2$ values to be computed.

- Parallel composition
  Let G' be constructed from G_1 and G_2 where s_1 and s_2 as well as t_1 and t_2 are identified. Assume that G_1 and G_2 and the respective J_{G_1,s_1}, J_{G_1,t_1}, J_{G_2,s_2}, J_{G_2,t_2}, W_{G_1} and W_{G_2} have been considered in a previous iteration.
  For the parallel composition, we compute a preliminary output $J'_{G',s}$, $J'_{G',t}$ and $W'_{G'}(a,b)$. The final output is computed in the subsequent cleansing step. We compute $J'_{G',s}$ and $J'_{G',t}$ by an element-wise addition (Minkowski sum): $J'_{G',s} \coloneqq J_{G_1,s_1} + J_{G_2,s_2}$ and $J'_{G',t} \coloneqq J_{G_1,t_1} + J_{G_2,t_2}$. For each element in $J'_{G',s} \times J'_{G',t}$, the value function is defined as follows:

$$W'_{G'}(a,b) \coloneqq \begin{cases} \max\Big\{W_{G_1}(a_1,b_1) + W_{G_2}(a_2,b_2)\Big| & (a_1,b_1) \in J_{G_1,s_1} \times J_{G_1,t_1} \\ & \wedge (a_2,b_2) \in J_{G_2,s_2} \times J_{G_2,t_2} \\ & \wedge a = a_1 + a_2 \\ & \wedge b = b_1 + b_2\Big\} \\ & \text{if } a \leq (1+\epsilon) b_s \wedge b \leq (1+\epsilon) b_t \\ -\infty & \text{else} \end{cases}$$

  By construction, for each tuple $(a,b) \in J'_{G',s} \times J'_{G',t}$ there is a pair of tuples $(a_1,b_1) \in J_{G_1,s_1} \times J_{G_1,t_1}$ and $(a_2,b_2) \in J_{G_2,s_2} \times J_{G_2,t_2}$, respectively, so that $a = a_1 + a_2$ and $b = b_1 + b_2$. So the set over which the maximum is determined, is non-empty and the above value is well-defined.
  We now consider the cleansing step that has to take place after each parallel composition. We first give the reasoning regarding why this is necessary: As stated before, for $G' = K_2$ it is clear that $\left|J_{G',s}\right| = \left|J_{G',t}\right| < (k+1)^2$. However, for the parallel composition

this no longer holds: If $\left|J_{G_1,s_1}\right| \leq (k+1)^2$ and $\left|J_{G_2,s_2}\right| \leq (k+1)^2$, the best upper bound we can give for $\left|J'_{G',s}\right|$ (or $\left|J'_{G',t}\right|$, respectively) as constructed above is $(k+1)^4$. A subsequent parallel composition might increase the number of stored values to $(k+1)^6$ and so on. Assuming a series parallel graph that is composed of $n-2$ parallel compositions of paths of length 2, we would have to store up to $(k+1)^{2(n-2)}$ values with the graph containing n nodes. Even if we neglected values larger than $(1+\epsilon)b_{\max}$ (since these cannot occur in the solution that we are looking for), we would store up to $\min\left\{(k+1)^{2(n-2)}, (1+\epsilon)b_{\max}\right\}$ values, which would yield a non-polynomial runtime. In order to avoid this, we reduce the values to be stored to at most $(k+1)^2$ representatives after each parallel composition. The cleansing step works as follows:
For each non-empty set $(J'_{G',s} \times J'_{G',t}) \cap (I_i \times I_j)$ we choose a representative tuple (a_i, b_j) for which the following condition holds:

$$W'_{G'}(a_i, b_j) \geq W'_{G'}(a,b) \text{ for all } (a,b) \in (J'_{G',s} \times J'_{G',t}) \cap (I_i \times I_j)$$

If there are multiple tuples satisfying this condition, an arbitrary representative is chosen.
We define the set $X_{G'}$ to be the union of all these representatives:

$$X_{G'} := \left\{(a_i, b_j) \,|\, i,j = 0,\ldots,k\right\}$$

From this set $X_{G'}$ we derive $J_{G',s}$ and $J_{G',t}$ using the following projections:

$$J_{G',s} := \{a \,|\, (a,b) \in X_{G'} \text{ for some b}\}$$

$$J_{G',t} := \{b \,|\, (a,b) \in X_{G'} \text{ for some a}\}$$

The set $X_{G'}$ contains at most $(k+1)^2$ values, which implies that $J_{G',s}$ and $J_{G',t}$ also contain at most $(k+1)^2$ values as claimed. The value function can be restricted to the relevant definition range: $W_{G'} := W'_{G'|J_{G',s} \times J_{G',t}}$. For the subsequent analysis it is worth noting that if J_{G_1,s_1}, J_{G_1,t_1}, J_{G_2,s_2} and J_{G_2,t_2} contain the value 0, then this also holds for $J'_{G',s}$ and $J'_{G',t}$. Since $I_0 = \{0\}$, this implies that $J_{G',s}$ and $J_{G',t}$ also contain the value 0.

- Series composition
  Let G' be constructed from G_1 and G_2 where t_1 and s_2 are identified.
  As before, we assume that G_1 and G_2 and the respective J_{G_1,s_1}, J_{G_1,t_1}, J_{G_2,s_2}, J_{G_2,t_2}, W_{G_1} and W_{G_2} have been considered in a previous iteration.
  Set $J_{G',s} := J_{G_1,s_1}$ and $J_{G',t} := J_{G_2,t_2}$. Trivially, this implies $\left|J_{G',s}\right| = \left|J_{G_1,s_1}\right|$ and $\left|J_{G',t}\right| =$

$\left|J_{G_2,t_2}\right|$. Recall that for $G' = K_2$ and for the parallel composition we have already shown that the cardinalities of the sets $J_{G',s}$ and $J_{G',t}$, that are produced as the output of the respective iteration, are bounded from above by $(k+1)^2$. By construction, this bound inductively also holds for the series composition. Thus, we can conclude that after each iteration of the algorithm the cardinalities of $J_{G',s}$ and $J_{G',t}$ are bounded from above by $(k+1)^2$. This result is an important input for the subsequent complexity analysis. The value function regarding G' is given by

$$W_{G'}(a,b) :=$$
$$\max\left\{W_{G_1}(a,b_1) + W_{G_2}(a_2,b)\,\middle|\, b_1 \in J_{G_1,t_1} \wedge a_2 \in J_{G_2,s_2} \wedge b_1 + a_2 \in \{0\} \cup \left[\frac{q_{t_1}}{1+\epsilon}, (1+\epsilon)b_{t_1}\right]\right\}$$

Note that t_1 and s_2 are the same node in G' so that it is sufficient to check the feasibility of $b_1 + a_2$ regarding q_t and b_t. The value $W_{G'}(a,b)$ is well-defined for each $(a,b) \in J_{G',s} \times J_{G',t}$: In order for the above set, over which the maximum is determined, to be non-empty, we have to show that there is a pair of values $(a_2,b_1) \in J_{G_1,t_1} \times J_{G_2,s_2}$ for which $b_1 + a_2 \in \{0\} \cup \left[\frac{q_{t_1}}{1+\epsilon}, (1+\epsilon)b_{t_1}\right]$.
In order to show this, we first have to prove that the value 0 is contained in $J_{G',s}$ and $J_{G',t}$ after each iteration. For the parallel composition we have already seen that if the input sets J_{G_1,s_1}, J_{G_1,t_1}, J_{G_2,s_2} and J_{G_2,t_2} of the underlying graphs contain the value 0, then so do $J_{G',s}$ and $J_{G',t}$. For the series composition we get by construction that if J_{G_1,s_1} and J_{G_2,t_2} contain 0, then this also holds for $J_{G',s}$ and $J_{G',t}$. As the composition of series parallel graphs starts on K_2, where we define $J_{G',s}$ and $J_{G',t}$ to contain 0 we inductively obtain the claim.
We conclude that the above maximum is created over a non-empty set, as it contains at least the value $W_{G_1}(a,0) + W_{G_2}(0,b)$. Furthermore, by construction, for all tuples (a,b) to be considered, $a \in J_{G_1,s_1}$ and $b \in J_{G_2,t_2}$, which implies that $W_{G_1}(a,b_1) + W_{G_2}(a_1,b)$ is always well-defined.

We continue with the above dynamic program until $G' = G$. The approximate value we are looking for and which we denote by ALG is determined as follows:

$$\begin{aligned}\mathrm{ALG} := \max\Big\{W_G(a,b)\,\Big|\,&(a,b) \in J_{G,s} \times J_{G,t}\\ &\wedge a \in \{0\} \cup \left[\frac{q_s}{1+\epsilon}, (1+\epsilon)b_s\right]\\ &\wedge b \in \{0\} \cup \left[\frac{q_t}{1+\epsilon}, (1+\epsilon)b_t\right]\Big\}\end{aligned}$$

Since the tuple $(0,0)$ is always contained in the set over which the maximum is determined, this value is well-defined.
Note that $W_{G'}(a,b)$ is in fact the value of the matching that corresponds to the tuple (a,b) and that the algorithm has stored. For K_2 this is obvious. During the parallel and series compositions we combine two edge-disjoint matchings and add their values, which is reflected in the computation of $W_{G'}$. So inductively, the claim follows.
Having outlined the dynamic program, we are now ready to show the theorem in three steps:

1. ALG $\geq$ OPT, where OPT denotes the optimal value of the given instance.

2. If m denotes the matching computed by the dynamic program, then for all $v \in V(G)$ we have that $m(\delta(v)) \in \{0\} \cup \left[\frac{q_v}{1+\epsilon}, (1+\epsilon)b_v\right]$. Furthermore, for all $e \in E(G)$ we have that $m_e \in \{0\} \cup \left[\frac{q_e}{1+\sigma}, b_e\right]$. Hence, the constraints on the nodes and edges are violated by at most a factor $1+\epsilon$.

3. The dynamic program runs in polynomial time.

ad 1. We introduce the following notation for the subgraphs G_i that occurs during the series parallel composition of G:

   - w_{G_i} denotes the weight-vector w restricted to the edges of G_i.
   - m^{OPT} denotes an (arbitrary but fixed) optimal solution to the given instance of MWECBMMQ.
   - $m^{\mathrm{OPT}}_{G_i}$ denotes the restriction of m^{OPT} to the edges of G_i.
   - $m^{\mathrm{ALG}}_{G_i}$ denotes the matching corresponding to a certain partial solution that the algorithm computes for G_i.
   - $\deg_{G_i}(v)$ for some $v \in V(G_i)$ denotes the degree of node v in G_i.

   We show by induction that for each G' the algorithm stores at least one solution $m^{\mathrm{ALG}}_{G'}$ corresponding to a tuple $(a,b) \in J_{G',s} \times J_{G',t}$, for which the following holds:

   a) $W_{G'}(a,b) \geq m^{\mathrm{OPT}}_{G'} w_{G'}$

   b) For $v \in \{s,t\}$ the following holds:

$$\frac{m^{\mathrm{OPT}}_{G'}(\delta(v))}{(1+\sigma)^{\deg_{G'}(v)}} \leq m^{\mathrm{ALG}}_{G'}(\delta(v)) \leq (1+\sigma)^{\deg_{G'}(v)} m^{\mathrm{OPT}}_{G'}(\delta(v))$$

   Note that if (a,b) denotes the tuple corresponding to $m^{\mathrm{ALG}}_{G_i}$, we always have $m^{\mathrm{ALG}}_{G_i}(\delta(s)) = a$ and $m^{\mathrm{ALG}}_{G_i}(\delta(t)) = b$.

The algorithm always starts with some graph $G' = K_2$, which is the base case of the induction. We have $\deg(s) = \deg(t) = 1$. In this case $J_{G',s} = J_{G',t} = \{p_i \mid i = 0, \ldots, k+1\}$. If e denotes the edge of G', we have $m_e^{\mathrm{OPT}} \in I_i$ for some i.
If $i = 0$ (i.e. $m_e^{\mathrm{OPT}} = 0$), then $m_e^{\mathrm{ALG}} = p_0$ fulfills a) and b).
Else, $q_e \leq m_e^{\mathrm{OPT}}$. We differentiate the following cases:

- If $m_e^{\mathrm{OPT}} = p_i$, then $m_e^{\mathrm{ALG}} = p_i$ fulfills a) and b).
- If $m_e^{\mathrm{OPT}} \geq p_i + 1$ and $w_e \geq 0$ then $m_e^{\mathrm{ALG}} = p_{i+1}$ fulfills the claim: Note that $m_e^{\mathrm{OPT}} \leq m_e^{\mathrm{ALG}} = p_{i+1} \leq (1+\sigma) m_e^{\mathrm{OPT}}$, which yields b). Since m_e^{OPT} is feasible, we have that $m_e^{\mathrm{OPT}} \leq \min\{b_s, b_t\}$ and $m_e^{\mathrm{OPT}} \in [q_e, b_e]$. Hence, $p_{i+1} \leq \min\{b_s, b_t\} \cdot (1+\sigma)$ and $p_{i+1} \in [q_e, (1+\sigma) b_e]$. This implies $W_{G'}(p_{i+1}, p_{i+1}) = p_{i+1} w_e \geq m_e^{\mathrm{OPT}} w_{G'} = m_e^{\mathrm{OPT}} w_e$, i.e. property a).
- If $m_e^{\mathrm{OPT}} \geq p_i + 1$ and $w_e < 0$ then $m_e^{\mathrm{ALG}} = p_i$ fulfills the claim: We have that $\frac{m_e^{\mathrm{OPT}}}{1+\sigma} \leq m_e^{\mathrm{ALG}} = p_i < m_e^{\mathrm{OPT}}$, which implies property b). By assumption, m_e^{OPT} is feasible. Thus, we have that $m_e^{\mathrm{OPT}} \leq \min\{b_s, b_t\}$ and $m_e^{\mathrm{OPT}} \in [q_e, b_e]$. Hence, $p_i \leq \min\{b_s, b_t\}$ and $p_i \in \left[\frac{q_e}{1+\sigma}, b_e\right]$. As before, we have that $W_{G'}(p_i, p_i) = p_i w_e > m_e^{\mathrm{OPT}} w_{G'} = m_e^{\mathrm{OPT}} w_e$, which implies property a).

For the induction step we have to consider the parallel and the series composition:
We consider the series composition first. As before, G' is constructed from G_1 and G_2 where t_1 and s_2 are identified. We assume that the properties a) and b) hold for all previous iterations. Thus, there are (at least) two solutions $m_{G_1}^{\mathrm{ALG}}$ and $m_{G_2}^{\mathrm{ALG}}$, for which a) and b) hold. We assume that these solutions correspond to the tuples $(a_1, b_1) \in J_{G_1,s_1} \times J_{G_1,t_1}$ and $(a_2, b_2) \in J_{G_2,s_2} \times J_{G_2,t_2}$. Since $J_{G',s} = J_{G_1,s_1}$ and $J_{G',t} = J_{G_2,t_2}$, we have that $(a_1, b_2) \in J_{G',s} \times J_{G',t}$. We show that $m_{G'}^{\mathrm{ALG}}$ corresponding to (a_1, b_2) fulfills a) and b). We already know that the following holds:

$$W_{G_1}(a_1, b_1) + W_{G_2}(a_2, b_2) \geq m_{G_1}^{\mathrm{OPT}} w_{G_1} + m_{G_2}^{\mathrm{OPT}} w_{G_2} = m_{G'}^{\mathrm{OPT}} w_{G'} \tag{1}$$

The inequality follows from the assumption that a) holds for the results from previous iterations. Since $m_{G'}^{\mathrm{OPT}}$ is the union of the edge-disjoint matchings $m_{G_1}^{\mathrm{OPT}}$ and $m_{G_2}^{\mathrm{OPT}}$, the equality also holds. In order to show property a), it is sufficient to prove that $W_{G_1}(a_1, b_1) + W_{G_2}(a_2, b_2)$ is contained in the set over which the maximum is determined during the calculation of $W_{G'}(a_1, b_2)$. We show that $a_2 + b_1 \in \{0\} \cup \left[\frac{q_{t_1}}{1+\epsilon}, (1+\epsilon) b_{t_1}\right]$, which yields the claim. By assumption, the following equations hold:

$$\frac{m_{G_1}^{\mathrm{OPT}}(\delta(t_1))}{(1+\sigma)^{\deg_{G_1}(t_1)}} \leq m_{G_1}^{\mathrm{ALG}}(\delta(t_1)) = b_1 \leq (1+\sigma)^{\deg_{G_1}(t_1)} m_{G_1}^{\mathrm{OPT}}(\delta(t_1)) \tag{2}$$

$$\frac{m_{G_2}^{\mathrm{OPT}}(\delta(t_1))}{(1+\sigma)^{\deg_{G_2}(t_1)}} \leq m_{G_2}^{\mathrm{ALG}}(\delta(t_1)) = a_2 \leq (1+\sigma)^{\deg_{G_2}(t_1)} m_{G_2}^{\mathrm{OPT}}(\delta(t_1)) \tag{3}$$

In the second equation we have used the fact that $t_1 = s_2$. Adding (2) and (3) and using the fact that $\deg_{G_1}(t_1) < n$ and $\deg_{G_2}(t_1) < n$ yields

$$\frac{m_{G_1}^{\mathrm{OPT}}(\delta(t_1)) + m_{G_2}^{\mathrm{OPT}}(\delta(t_1))}{(1+\sigma)^n} \leq b_1 + a_2 \leq (1+\sigma)^n (m_{G_1}^{\mathrm{OPT}}(\delta(t_1)) + m_{G_2}^{\mathrm{OPT}}(\delta(t_1)))$$

By definition, $(1+\sigma)^n = 1+\epsilon$ and $m_{G_1}^{\mathrm{OPT}}(\delta(t_1)) + m_{G_2}^{\mathrm{OPT}}(\delta(t_1)) = m_{G'}^{\mathrm{OPT}}(\delta(t_1))$.
As a result we get

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(t_1))}{1+\epsilon} \leq b_1 + a_2 \leq (1+\epsilon) m_{G'}^{\mathrm{OPT}}(\delta(t_1))$$

Since $m_{G'}^{\mathrm{OPT}}$ is a feasible solution in G', we have that $m_{G'}^{\mathrm{OPT}}(\delta(t_1)) \in \{0\} \cup [q_{t_1}, b_{t_1}]$ and thus also $a_2 + b_1 \in \{0\} \cup \left[\frac{q_{t_1}}{1+\epsilon}, (1+\epsilon) b_{t_1}\right]$. We have established the fact that the term $W_{G_1}(a_1, b_1) + W_{G_2}(a_2, b_2)$ is considered when calculating $W_{G'}(a_1, b_2)$, so we obtain $W_{G'}(a_1, b_2) \geq W_{G_1}(a_1, b_1) + W_{G_2}(a_2, b_2)$. Together with (1), this yields property (a).
By assumption we have

$$\frac{m_{G_1}^{\mathrm{OPT}}(\delta(s_1))}{(1+\sigma)^{\deg_{G_1}(s_1)}} \leq m_{G_1}^{\mathrm{ALG}}(\delta(s_1)) \leq (1+\sigma)^{\deg_{G_1}(s_1)} m_{G_1}^{\mathrm{OPT}}(\delta(s_1))$$

$$\frac{m_{G_2}^{\mathrm{OPT}}(\delta(t_2))}{(1+\sigma)^{\deg_{G_2}(t_2)}} \leq m_{G_2}^{\mathrm{ALG}}(\delta(t_2)) \leq (1+\sigma)^{\deg_{G_2}(t_2)} m_{G_2}^{\mathrm{OPT}}(\delta(t_2))$$

We use the fact that node s_1 in G_1 equals node s in G' and that $\deg_{G_1}(s_1) = \deg_{G'}(s_1)$. The same argument holds for t_2 and t. We obtain the following:

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\deg_{G'}(s)}} \leq m_{G'}^{\mathrm{ALG}}(\delta(s)) \leq (1+\sigma)^{\deg_{G'}(s)} m_{G'}^{\mathrm{OPT}}(\delta(s))$$

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(t))}{(1+\sigma)^{\deg_{G'}(t)}} \leq m_{G'}^{\mathrm{ALG}}(\delta(t)) \leq (1+\sigma)^{\deg_{G'}(t)} m_{G'}^{\mathrm{OPT}}(\delta(t))$$

So we have also established property (b).

We now consider the parallel composition. By assumption, there are two solutions $m_{G_1}^{\mathrm{ALG}}$ and $m_{G_2}^{\mathrm{ALG}}$ and the corresponding tuples (a_1, b_1) and (a_2, b_2) that fulfill a) and b). By construction, for the graph G' the tuple $(a_1 + a_2, b_1 + b_2)$ is considered by the algorithm before the cleansing step, i.e. $W'_{G'}(a_1 + a_2, b_1 + b_2)$ is determined. We

show that by the end of the iteration, a solution and the corresponding tuple $(a,b) \approx (a_1+a_2, b_1+b_2)$ that fulfill a) and b), are stored by the algorithm.
As for the series composition, it is clear that (1) also holds for the parallel composition. We begin by proving that $W'_{G'}(a_1+a_2, b_1+b_2) \geq W_{G_1}(a_1,b_1) + W_{G_2}(a_2,b_2)$. In order to do so, it is sufficient to show that the first case of the case differentiation for determining $W'_{G'}(a_1+a_2, b_1+b_2)$ applies, i.e. $a_1+a_2 \leq (1+\epsilon)b_s$ and $b_1+b_2 \leq (1+\epsilon)b_t$. Then the set over which the maximum is determined contains $W_{G_1}(a_1,b_1) + W_{G_2}(a_2,b_2)$. By assumption and taking into account that s_1 in G_1, s_2 in G_2 and s in G' are the same node, the following holds:

$$\frac{m_{G_1}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\deg_{G_1}(s)}} \leq a_1 = m_{G_1}^{\mathrm{ALG}}(\delta(s)) \leq (1+\sigma)^{\deg_{G_1}(s)} m_{G_1}^{\mathrm{OPT}}(\delta(s))$$

$$\frac{m_{G_2}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\deg_{G_2}(s)}} \leq a_2 = m_{G_2}^{\mathrm{ALG}}(\delta(s)) \leq (1+\sigma)^{\deg_{G_2}(s)} m_{G_2}^{\mathrm{OPT}}(\delta(s))$$

Summation of these terms and using $m_{G_1}^{\mathrm{OPT}}(\delta(s)) + m_{G_2}^{\mathrm{OPT}}(\delta(s)) = m_{G'}^{\mathrm{OPT}}(\delta(s))$ yields:

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\max\{\deg_{G_1}(s), \deg_{G_2}(s)\}}} \leq a_1 + a_2 \leq (1+\sigma)^{\max\{\deg_{G_1}(s), \deg_{G_2}(s)\}} m_{G'}^{\mathrm{OPT}}(\delta(s)) \quad (4)$$

We only consider the second inequality of (4) and use the fact that the degree of the nodes in G is bounded from above by n:

$$a_1 + a_2 \leq (1+\epsilon) m_{G'}^{\mathrm{OPT}}(\delta(s)) \leq (1+\epsilon) b_s$$

Analogously we get that $b_1 + b_2 \leq (1+\epsilon)b_t$. Using (1), we have shown the following:

$$W'_{G'}(a_1+a_2, b_1+b_2) \geq m_{G'}^{\mathrm{OPT}} w_{G'}$$

Once the first part of the iteration has been completed for all tuples in $J'_{G',s} \times J'_{G',t}$, the cleansing step follows. By construction, a representative (a,b) from $J'_{G',s} \times J'_{G',t}$ is stored in $J_{G',s} \times J_{G',t}$ for which $W_{G'}(a,b) = W'_{G'}(a,b) \geq W'_{G'}(a_1+a_2, b_1+b_2)$. So (a,b) fulfills a).

Furthermore, $(a,b) \approx (a_1+a_2, b_1+b_2)$ or more specifically:

$$\frac{a_1+a_2}{1+\sigma} \leq a \leq (1+\sigma)(a_1+a_2) \tag{5}$$

$$\frac{b_1+b_2}{1+\sigma} \leq b \leq (1+\sigma)(b_1+b_2)$$

We combine (4) and (5):

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\max\{\deg_{G_1}(s),\deg_{G_2}(s)\}+1}} \leq \frac{a_1+a_2}{1+\sigma} \leq a \leq (1+\sigma)(a_1+a_2) \leq (1+\sigma)^{\max\{\deg_{G_1}(s),\deg_{G_2}(s)\}+1} m_{G'}^{\mathrm{OPT}}(\delta(s))$$

Note that $\max\{\deg_{G_1}(s), \deg_{G_2}(s)\}+1 \leq \deg_{G_1}(s)+\deg_{G_2}(s) = \deg_{G'}(s)$, so we obtain

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(s))}{(1+\sigma)^{\deg_{G'}(s)}} \leq a \leq (1+\sigma)^{\deg_{G'}(s)} m_{G'}^{\mathrm{OPT}}(\delta(s))$$

By the same argument we get

$$\frac{m_{G'}^{\mathrm{OPT}}(\delta(t))}{(1+\sigma)^{\deg_{G'}(t)}} \leq b \leq (1+\sigma)^{\deg_{G'}(t)} m_{G'}^{\mathrm{OPT}}(\delta(t))$$

Recall that $a = m_{G'}^{\mathrm{ALG}}(\delta(s))$ and $b = m_{G'}^{\mathrm{ALG}}(\delta(t))$. So we have also shown property b). This completes the induction.

Property 1, i.e. the quality of the approximation, now follows in a straightforward way: By the above induction, properties a) and b) hold for the graph $G' = G$. Hence, there is a tuple $(a,b) \in J_{G,s} \times J_{G,t}$ for which the following holds:

- $W_G(a,b) \geq m_G^{\mathrm{OPT}} w_G$
- $\frac{q_s}{1+\epsilon} \leq a \leq (1+\epsilon) b_s$ or $a = 0$
- $\frac{q_t}{1+\epsilon} \leq b \leq (1+\epsilon) b_t$ or $b = 0$

The second and the third statement follow from property b), using that $\deg_G(s)$ and $\deg_G(t)$ are bounded from above by n (G is a simple graph), m_G^{OPT} is feasible, $m_G^{\mathrm{ALG}}\delta((s)) = a$ and $m_G^{\mathrm{ALG}}\delta((t)) = b$. This implies that during the computation of the value ALG as defined above $W_G(a,b)$ is considered. Thus, $\mathrm{ALG} \geq W_G(a,b) \geq m_G^{\mathrm{OPT}} w_G = \mathrm{OPT}$.

ad 2. We show inductively that for all tuples (a,b) for which $W_{G'}(a,b) > -\infty$ the corresponding matching $m_{G'}^{\mathrm{ALG}}$ fulfills

$$m_{G'}^{\mathrm{ALG}}(\delta(v)) \in \{0\} \cup \left[\frac{q_v}{1+\epsilon}, (1+\epsilon)b_v\right] \text{ for all v } \in V(G') \setminus \{s,t\} \tag{6}$$

As the base case we consider $G' = K_2$. In this case $V(G') \setminus \{s,t\}$ is empty, so the claim always holds. For the induction step, we show that the parallel and series composition maintain this property.

We consider the series composition first. Assume that $W_{G'}(a,b) > -\infty$. By definition we have that $W_{G'}(a,b) = W_{G_1}(a,b_1) + W_{G_2}(a_1,b)$ for some $b_1 \in J_{G_1,t_1}$ and $a_2 \in J_{G_2,s_2}$. This implies that $W_{G_1}(a,b_1) > -\infty$ and $W_{G_2}(a_1,b) > -\infty$. So, by assumption, the corresponding matchings fulfill (6). Hence the only non-terminal node in G', for which we have to show (6), is $t_1 = s_2$. By definition of $W_{G'}(a,b)$, we have that $m_{G'}^{\mathrm{ALG}}(\delta(t_1)) = b_1 + a_2 \in \{0\} \cup \left[\frac{q_{t_1}}{1+\epsilon}, (1+\epsilon)b_{t_1}\right]$, which yields the claim.

For the parallel composition we assume again that $W_{G'}(a,b) > -\infty$. We have that $W_{G'}(a,b) = W_{G_1}(a_1,b_1) + W_{G_2}(a_2,b_2)$ for some $(a_1,b_1) \in J_{G_1,s_1} \times J_{G_1,t_1}$ and $(a_2,b_2) \in J_{G_2,s_2} \times J_{G_2,t_2}$. Once again, we have that $W_{G_1}(a_1,b_1) > -\infty$ and $W_{G_2}(a_2,b_2) > -\infty$, so (6) holds for the corresponding matchings in G_1 and G_2. For the parallel composition, the set of non-terminal nodes in G' is exactly union of the sets of non-terminal nodes in G_1 and G_2 ($s = s_1 = s_2$ and $t = t_1 = t_2$). So (6) also holds for G'.

Using the lower bound for ALG, that we have just shown in 1., and taking into account that we can always choose an empty matching as a trivial feasible solution with value 0, we get that $\mathrm{ALG} \geq \mathrm{OPT} \geq 0 > -\infty$ always holds. By construction, $\mathrm{ALG} = W_G(a,b)$ for some (a,b). Applying the result of the above induction to the corresponding matching m, we get that (6) holds for all non-terminal nodes of G. By construction of ALG, the constraints of the terminal nodes are violated by at most a factor $1+\epsilon$. Hence we obtain $m(\delta(v)) \in \{0\} \cup \left[\frac{q_v}{1+\epsilon}, (1+\epsilon)b_v\right]$ for all $v \in V(G)$.

As we have just seen, $\mathrm{ALG} > -\infty$. Assume that for the corresponding matching m we had $m_e \notin \{0\} \cup \left[\frac{q_e}{1+\sigma}, (1+\sigma)b_e\right]$ for some $e \in E(G)$: At some point the algorithm considers the graph $G' = K_2$, consisting of e and its end nodes. Then we would have $W_{G'}(m_e,m_e) = -\infty$. By construction of the algorithm we inductively get that this would also imply $\mathrm{ALG} = -\infty$, which yields a contradiction. Hence, $m_e \in \{0\} \cup \left[\frac{q_e}{1+\sigma}, (1+\sigma)b_e\right]$ for all $e \in E(G)$.

ad 3. We now show that the algorithm runs in polynomial time. As an input we assume a series parallel graph G as well as a sequence of series and parallel compositions for constructing G. Given a series parallel graph, such a sequence can be computed in

polynomial time (Lemma 1.4.2).

Once a sequence of compositions has been determined, the algorithm can be applied. We first consider the computations regarding the one-edge graphs K_2. The number of such graphs to be considered is polynomially bounded in the encoding size of G, as there are at most $|E(G)|$ such graphs. For each of these graphs, at most $(k+2)^2$ values are computed in polynomial time. As already stated, k is polynomial in the encoding size of the given instance of MWECBMMQ.

Once all base graphs K_2 have been considered by the algorithm, the graph G can be constructed iteratively by series and parallel compositions. Note that according to Lemma 1.4.2 the sequence of series and parallel compositions can be represented as a binary sp-tree, where the leaf nodes correspond to the one-edge graphs. Hence, the number of non-leaf nodes, i.e. the number of series and parallel compositions, is bounded by $O(|E(G)|)$. Recall that we have already established the fact that $\left|J_{G',s}\right|$ and $\left|J_{G',t}\right|$ are bounded from above by $(k+1)^2$ for each subgraph G'.

For each iteration in which a series composition is considered we have to compute $W_{G'}$ for at most $(k+1)^4$ tuples. Each of these computations consists of computing the maximum over a set with at most $(k+1)^4$ elements. All in all, the computational complexity of each such iteration is in $O(k^8)$.

For each parallel composition we have to compute $W'_{G'}$ for up to $(k+1)^8$ tuples ($J'_{G',s}$ and $J'_{G',t}$ both have up to $(k+1)^4$ elements). In this case, a maximum over a set with up to $(k+1)^8$ elements has to be computed, which yields an overall computational complexity of $O(k^{16})$ of this step. Once all values for $W'_{G'}$ have been computed, the cleansing step takes place. Each set $(J'_{G',s} \times J'_{G',t}) \cap (I_i \times I_j)$ contains at most $(k+1)^8$ values. Thus, finding a representative maximizing $W'_{G'}$ for each of these sets can be achieved in $O(k^8)$ and there are $(k+1)^2$ such sets, which yields a complexity of $O(k^{10})$. The subsequent projection of the tuples can be done in $O(k^2)$. So all in all, we get that the computational complexity of each such iteration is in $O(k^{16})$.

After all iterations have been completed, ALG is determined by computing the maximum over a set with at most $(k+1)^4$ elements, which can be done in $O(k^4)$. All in all we obtain the computational complexity $O(k^{16}|E(G)|)$.

Note that according to Definition 1.3.9, an (α, β)-approximation has to return infeasibility of the given instance if there is in fact no feasible solution to the instance. However, since an empty matching is a feasible solution to every instance of MWECBMMQ, infeasibility can never occur. □

4.4 Conclusion

In this chapter we have generalized the notion of maximum-weight b-matchings by applying the concept of minimum quantities (Definition 4.2.3) and we have called this problem "maximum-weight b-matching problem with minimum quantities" (MWBMMQ). As expected, this generalization makes a polynomially solvable problem NP-hard.
Specifically, the following variant of MWBMMQ is strongly NP-hard:

- MWBMMQ on bipartite graphs with uniform edge weights (Corollary 4.3.2)
  On the other hand, there is no polynomial approximation algorithm unless P = NP (Theorem 4.3.4).

We could show at least weak NP-hardness of (even approximating) the following problems:

- MWBMMQ on series parallel graphs with uniform edge weights (Theorem 4.3.4 and Corollary 4.3.17)
  A polynomial $(1,1+\epsilon)$-approximation exists even for MWECBMMQ (Theorem 4.3.18)
- MWBMMQ on trees with uniform edge weights (Corollary 4.3.5 and Corollary 4.3.17)
- MWBMMQ on complete binary trees (Theorem 4.3.6 and Corollary 4.3.17)

Note that according to Remark 4.3.7, MWECBMMQ is at least as hard as MWBMMQ.
Furthermore, we could show that given a value $\epsilon > 0$ for which it is strongly NP-hard to decide whether an instance of 3DM-3 has a solution with value n or whether all feasible solutions have value at most $(1-\epsilon)n$, this also holds for MWBMMQ. The statement even holds on bipartite graphs with $w_e = 1$ on all edges $e \in E$ and $q_v = b_v \leq 3$ for all $v \in V$ (Theorem 4.3.1).
Conversely, we could show polynomial solvability for the following variants:

- MWBMMQ on graphs where $V(G) \leq k$ for $k \in \mathbb{N}$ fixed (Theorem 4.3.8)
- MWBMMQ on paths (Theorem 4.3.9)
- MWBMMQ on cycles (Corollary 4.3.10)
- MWBMMQ on graphs where $b_v \leq 2$ for all $v \in V$ (Theorem 4.3.11)

In the following case, we could show pseudo-polynomial solvability:

- MWECBMMQ on graphs for which the treewidth is at most t for fixed $t \geq 1$ (Theorem 4.3.12)

From the previous result we concluded that the following problem can be solved in linear time:

- MWECBMMQ on graphs where the treewidth *and* the maximum capacity are both bounded from above (Corollary 4.3.16)

Furthermore, we could show linear-time solvability for the following problem defined in Chapter 2:

- RSAP (and RPMP) where the treewidth of the corresponding feasibility graph and the maximum bin capacity are bounded by constants (Corollary 4.3.15)

Chapter 5

Flow problems

5.1 Introduction

Flow problems are another natural candidate for applying minimum quantities: As pointed out in [25] and [45], flow problems with minimum quantities could, for example, be applied to water supply and sewerage networks. If the rate of flow through the pipes or sewers is too low, this might lead to stagnating fresh water being contaminated over time or the sewers being clogged. Hence, it might be desirable to determine a flow so that edges are either unused or there is at least a certain rate of flow on them. It might be due to these obvious use cases that, among the classes of optimization problems with minimum quantities that are within the scope of this thesis, most previous publications focus on flow problems with minimum quantities.
In the next section we give an overview of these publications and provide the formal definitions. In the subsequent section we contribute additional results regarding the minimum-cost flow problem with minimum quantities and the maximum flow problem with minimum quantities. Furthermore, we consider *generalized* flow problems with minimum quantities and analyze the relationship between non-generalized flow problems with minimum quantities and generalized flow problems with minimum quantities.

5.2 Basics and Definitions

We begin this chapter by providing the relevant definitions that we use in the following. Unless specified otherwise, we assume that the graphs in this chapter are simple, i.e. there are no loops or parallel edges. Given a graph $G = (V, E)$ and a flow $f : E \to \mathbb{N}$, we use the following notation: We denote the flow on some edge $e = (v_1, v_2) \in E$ by f_e. Since the graphs

in this chapter are usually simple, every edge can be identified by its end points. This allows us to use the alternative notation $f(v_1, v_2) = f_e$. For a set of edges $E' \subseteq E$ we use the notation $f(E') \coloneqq \sum_{e \in E'} f_e$.
First of all, we give a generic definition of flow problems with minimum quantities. In order to make the definition as comprehensive as possible, we already incorporate the notion of *generalized* flows. In a generalized flow network, the input of an edge does not necessarily equal its output, i.e. there might be a gain or a loss regarding the input. This is represented by a multiplier on each edge. More information on generalized flow problems can be found in [1] and [34]. Furthermore, the authors of [48] have already considered a similar version of the generalized minimum cost flow problem with minimum quantities.

Definition 5.2.1 (generalized flow network with minimum quantities)**.** A generalized flow network with minimum quantities is given by a directed graph $G = (V, E)$, where V denotes the set of nodes and E denotes the set of directed edges. There are two dedicated nodes s and t that we refer to as the source and the sink. Minimum quantities $q_e \in \mathbb{N}$ and upper capacities $b_e \in \mathbb{N}$ are defined for all $e \in E$. In addition, a multiplier $\mu_e \in \mathbb{Q}_+$ is defined on all edges. In order for a flow $f : E \to \mathbb{N}, e \mapsto f_e$ to be feasible, we require $\mu_e f_e \in \mathbb{N}$, $f_e \in \{0\} \cup [q_e, b_e]$ for all $e \in E$ and $\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} \mu_e f_e = 0$ for all $v \in V \setminus \{s, t\}$.
If the multipliers μ_e are uniformly one, we usually omit the multipliers as well as the prefix "generalized" and refer to the flow network as a "flow network with minimum quantities".

We allow edge capacities $b_e = 0$ for technical reasons. Trivially, adding or removing these edges does not affect the set of feasible flows.
There are different possibilities to define integrality of generalized flows. The authors of [34] propose the following variants:

- An integral amount of flow enters each edge.
- An integral amount of flow enters each edge and each node.
- An integral amount of flow enters and leaves each edge.

Our definition corresponds to the third variant which is the most restrictive one.

Remark 5.2.2. While some authors allow edges to point to the source or to start at the sink, we assume $\delta^-(s) = \delta^+(t) = \emptyset$ in the following. Note that we can make this assumption w.l.o.g: Given a flow network where $\delta^-(s) \neq \emptyset$ holds for the source node s, we introduce a new node s' and an edge $e' = (s', s)$. We set $q_{e'} \coloneqq 0$, $b_{e'} \coloneqq \sum_{e \in \delta^+(s)} b_e$ and (if edge costs are relevant) $c_{e'} \coloneqq 0$. If the flow network is a *generalized* flow network, we set $\mu_{e'} \coloneqq 1$. Furthermore

we add the constraint $\sum_{e\in\delta^+(s)} f_e - \sum_{e\in\delta^-(s)} f_e = 0$. We make s' the dedicated source of the new flow network for which $\delta^-(s') = \emptyset$ holds by construction. If $\delta^+(t) \neq \emptyset$ for the sink node t, we transform the flow network analogously. It is easy to see the equivalence of both flow networks.

In this thesis we mainly consider two flow problems with minimum quantities. The first one is a generalization of the minimum-cost flow problem:

Definition 5.2.3 (generalized minimum cost flow problem with minimum quantities)**.** The generalized minimum cost flow problem with minimum quantities (GMCFMQ) is given by a generalized flow network with minimum quantities as defined in Definition 5.2.1, edge costs $c_e \in \mathbb{Q}$ for all $e \in E$ and a flow value $F \in \mathbb{N}$. In order for a flow f to be feasible, it must fulfill the constraints given in Definition 5.2.1 and $\sum_{e\in\delta^+(s)} f_e = F$. The task is to find a feasible flow f that minimizes $\sum_{e\in E} c_e f_e$.
If the multipliers μ_e are uniformly one, we usually omit the multipliers and refer to the problem as the minimum cost flow problem with minimum quantities (MCFMQ).

The second problem we analyze is a variant of the maximum flow problem:

Definition 5.2.4 (generalized maximum flow problem with minimum quantities)**.** The generalized maximum flow problem with minimum quantities (GMFMQ) is given by a generalized flow network with minimum quantities as defined in Definition 5.2.1. In order for a flow f to be feasible, it must fulfill the constraints given in Definition 5.2.1. The task is to find a feasible flow f that maximizes $\sum_{e\in\delta^+(s)} f_e$, i.e. the amount of flow leaving s.
If the multipliers μ_e are uniformly one, we usually omit the multipliers and refer to the problem as the maximum flow problem with minimum quantities (MFMQ).

Problems similar to MCFMQ and MFMQ have already been analyzed in other publications: MCFMQ (or a similar problem) has been considered in [25, 31, 32, 44, 45], while MFMQ (or a similar problem) has been considered in [14, 23, 25, 47]. We now give a brief summary of these publications and point out if the definitions of the respective problems differ from the definitions we use. Unless specified otherwise, we summarize the other publications based on the terminology presented above, even if the terminology used by the authors differs.
A variation of MCFMQ was first defined in [44]: The authors allow minimum quantities, edge capacities and edge costs as well as edge flows to be (positive) real numbers instead of natural numbers, which is required according to the definition we use. Furthermore, minimum quantities are only defined on the edges starting at the source node. The problem

is shown to be NP-hard on series parallel graphs using a reduction from SUBSET SUM. The same reduction can also be used to show NP-hardness of MCFMQ. Besides proving the complexity of the problem, the authors provide a branch-and-bound algorithm and analyze its performance.

The problem MCFMQ is further analyzed in [31, 32]. The definition in these papers coincides with ours. In [31], strong NP-hardness of MCFMQ is shown. Furthermore, the authors prove that unless P=NP, there cannot be any polynomial-time $g(|I|)$-approximation for MCFMQ, where $|I|$ is the encoding length of a given instance of MCFMQ and g is an arbitrary polynomial function. Both results hold even on bipartite graphs and if a feasible solution is guaranteed to exist. In [32] the same authors show that unless P=NP, there cannot be any polynomial-time $g(|I|)$-approximation for MCFMQ on series parallel graphs.

In [45] a dynamic program for MCFMQ with pseudo-polynomial running time on series parallel graphs, which was initially given in [31], is shown to be flawed and is subsequently fixed. In addition, it is shown that unless P=NP, MCFMQ cannot be approximated in polynomial time even for uniform minimum quantities. The definition of MCFMQ in [45] is the same as the one used in this thesis.

Finally, the authors of [25] provide a pseudo-polynomial dynamic program for the special case of MCFMQ where the minimum quantities are uniform, which has a better complexity than the one covering the general case given in [45]. Furthermore, NP-hardness of MCFMQ with uniform minimum quantities on series parallel graphs and on extension-parallel graphs is shown.

In [23] a problem similar to MFMQ is introduced. The definition differs from ours, as the edge flows may attain positive real numbers. In addition, the definition includes a circulation edge from the sink to the source, which is simply a technical difference regarding the representation of the problem. A reduction from SUBSET SUM is used to show that the problem is NP-hard. The reduction given in the paper could also be used to show NP-hardness of MFMQ. In the following, the authors provide a mixed integer linear program formulation. An approach combining Lagrangean relaxations and variable fixing is suggested. In addition, a heuristic for finding an approximate solution is proposed and the performance of several variants of that heuristic is compared. It is shown that the quality of the solution computed by the heuristic decreases as the percentage of edges on which a non-trivial minimum quantity constraint is defined increases.

Another approximate heuristic, which is based on the Edmonds-Karp algorithm, is proposed in [14] and compared to the one given in [23]. It is shown that the algorithm proposed by the authors is able to find a positive flow in several graphs, where the algorithm from [23] failed to find a positive flow. However, the solution generated by the algorithm can be considerably

worse than an optimal solution. The authors of [14] use the same definition of the problem that was given in [23].

The authors of [47], who define MFMQ as we do, show strong NP-hardness of approximating MFMQ on bipartite graphs and NP-hardness of MFMQ with uniform minimum quantities. Furthermore, a pseudo-polynomial dynamic program for MFMQ on series parallel graphs is provided. A polynomial-time $(2 - \frac{1}{q})$-approximation algorithm is shown for the case that the minimum quantities are uniformly q. Finally, the authors show that MFMQ can be solved in polynomial time if the underlying graph is series parallel *and* the minimum quantities are uniform.

In [25], the authors apply minimum quantities to the multi-commodity maximum flow problem, where different commodities have to be sent through a flow network. For each commodity there is a dedicated source and a sink and the task is to maximize the sum of the flows for each commodity. The authors also consider the special case where there is just one commodity, which coincides with our definition of MFMQ. The authors provide an approximation algorithm for MFMQ with identical minimum quantities that is at least as good as the one given in [47]. If the minimum quantities fulfill certain conditions and the underlying graph is series parallel, the multi-commodity maximum flow problem with minimum quantities is shown to be solvable in linear time. Polynomial or even linear algorithms are provided for several special cases of MFMQ on series parallel graphs and on pearl graphs with uniform minimum quantities. On the other hand, it is shown that MFMQ is NP-hard on pearl graphs in general. Finally, the authors show that MFMQ is *strongly* NP-hard and unless P=NP, no polynomial-time $g(|I|)$-approximation exists.

The application of minimum quantity constraints to *generalized* flow problems has been considered in [48]: The authors define the "minimal-cost network flow problem with variable lower bounds" (MCNF-VLN). The definition varies from ours in several ways: The definition given in [48] also allows *fixed* lower bounds on the edges, while we only allow minimum quantity constraints. For the source node a *maximum* supply is given and for the sink node a *minimal* demand is defined. In our model of MCFMQ the demand of the sink node is unconstrained, while an *exact* supply is specified for the source node. The authors of [48] do not restrict the flow to integral values. A mixed integer linear programming (MILP) formulation of MCNF-VLN is given and MCNF-VLN is shown to be NP-hard. The remainder of the paper focusses on computational results.

As we have just seen, there is a variety of different definitions of flow problems with minimum quantities. Unless specified otherwise, in this thesis we consider the problems MCFMQ and MFMQ as defined in Definition 5.2.3 and Definition 5.2.4. However, in some cases we also consider the variant where the edge flow may attain positive *real* values (which also includes

omitting the constraint $\mu_e f_e \in \mathbb{N}$), while the remaining definition of the respective problem is unchanged. We refer to these variants as MCRFMQ and MRFMQ.
The following table summarizes the variants of flow problems with minimum quantities that we consider in this thesis:

Problem	Objective function	Integral flow?	Generalized flow?
GMFMQ	Maximum flow	Yes	Yes
MFMQ	Maximum flow	Yes	No
MRFMQ	Maximum flow	No	No
GMCFMQ	Minimum-cost flow	Yes	Yes
MCFMQ	Minimum-cost flow	Yes	No
MCRFMQ	Minimum-cost flow	No	No

In the beginning of this chapter, we have pointed out that unless specified otherwise, all graphs are assumed to be simple. Note that we can do so w.l.o.g.: If a graph contains a loop or a parallel edge, the edge can be replaced by a path of length two. If the minimum quantities, capacities and (if relevant) edge costs and multipliers on the edges of the path are chosen appropriately, this transformation does not affect the solution to the respective problem instance. The transformation can be carried out in polynomial time and the size of the new problem instance is polynomially bounded in the size of the initial problem instance.

5.3 Complexity Results & Algorithms

We begin this chapter by showing that every instance of MFMQ or MCFMQ can be reduced to a matching problem with minimum quantities:

Theorem 5.3.1. *Every instance of MFMQ or MCFMQ can be reduced to an instance of MWECBMMQ (Definition 4.2.6) in polynomial time.*

Proof. Let an instance I of MFMQ or MCFMQ on a directed graph $G = (V, E)$ be given. Let n denote the number of nodes and let k denote the number of edges. Set $B := \max_{e \in E} b_e + 1$.
We create an instance I' of MWECBMMQ as follows: Replace every node $v \in V \setminus \{s, t\}$ by v^- and v^+, set $\delta(v^-) := \delta^-(v)$, $\delta(v^+) := \delta^+(v)$ and introduce a new edge $e_v = (v^-, v^+)$. We denote the new set of nodes by V', the set of *additional* edges by E' and the new graph by $G' = (V', E \cup E')$. Set $q_e := 0$ and $b_e := kB$ for all $e \in E'$. For all $e \in E$ reuse the minimum quantities and capacity constraints from G. Set $q_v := b_v := kB$ for all $v \in V' \setminus \{s, t\}$.
If the original instance is an instance of MFMQ, set $w_e := 1$ for all $e \in \delta(s) = \delta^-(s)$ and $w_e := 0$ for all $e \in E \setminus \delta(s)$. Set $q_s := q_t := 0$ and $b_s := b_t := kB$.

If the original instance is an instance of MCFMQ, introduce an additional node s' and an edge $e_s = (s', s)$ in E'. Set $q_{e_s} := b_{e_s} := 1$. Set $w_e := -c_e$ for all $e \in E$, $w_{e_s} := kB\left(\max_{e \in E} |c_e| + 1\right)$ and $w_e := 0$ for all $e \in E' \setminus \{e_s\}$. Set $q_t := b_t := F$, $q_s := b_s := F + 1$ and $q_{s'} := b_{s'} := 1$.
We claim that we can solve I by solving I'.
Figure 5.3.1, Figure 5.3.2 and Figure 5.3.3 illustrate the transformation of an instance of MFMQ or MCFMQ, respectively, into an instance of MWECBMMQ. Let the following instance of MCFMQ be given and let the required flow value be F. Note that by dropping the flow requirement and the costs, it can be interpreted as an instance of MFMQ.

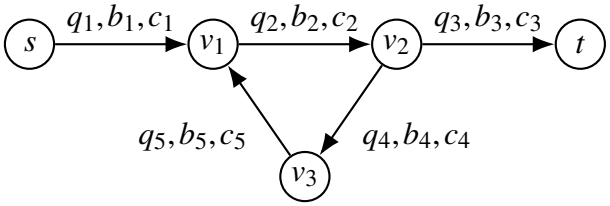

Fig. 5.3.1 Input instance MCFMQ / MFMQ; The edge labels represent the minimum quantities, capacities and costs

We obtain the following instance of MWECBMMQ from applying the transformation to the instance of MFMQ shown in Figure 5.3.1:

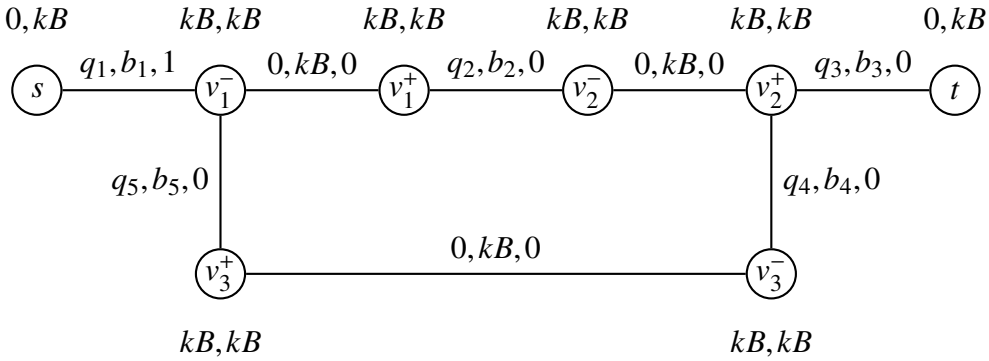

Fig. 5.3.2 Reducing MFMQ to MWECBMMQ, output instance; The edge labels represent the minimum quantities, capacities and weights on the respective edge; The node labels represent the minimum quantities and capacities on the respective node

Likewise, applying the transformation to the instance of MCFMQ shown in Figure 5.3.1 yields the following instance of MWECBMMQ:

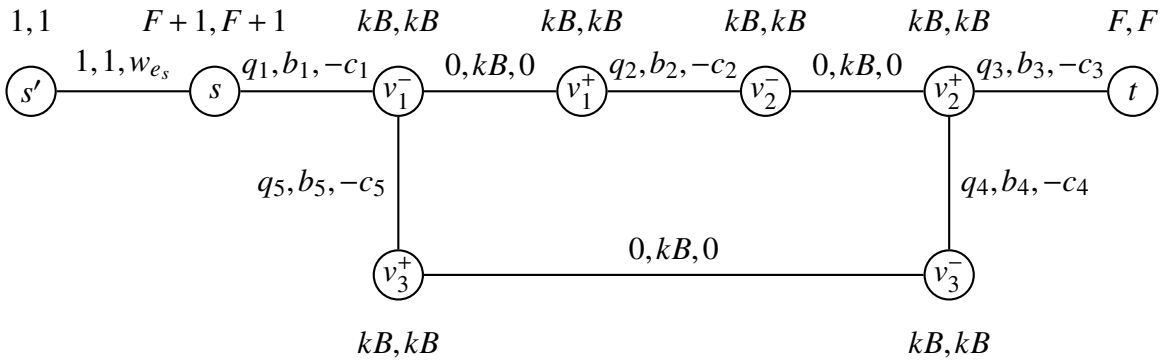

Fig. 5.3.3 Reducing MCFMQ to MWECBMMQ, output instance; The edge labels represent the minimum quantities, capacities and weights on the respective edge; The node labels represent the minimum quantities and capacities on the respective node

We now show that an optimal solution to the instance I' of MWECBMMQ as constructed above induces an optimal solution to the instance I of MCFMQ or MFMQ, respectively.
We first consider MFMQ. We show how a feasible solution to I can be transformed into a feasible solution to I' and vice versa. Let a feasible solution f to I be given and let f_e and m_e denote the flow or matching on some edge e. For all edges $e \in E \cup E'$ set

$$m_e := \begin{cases} f_e & \text{if } e \in E \\ kB - f(\delta^-(v)) & \text{if } e = (v^-, v^+) \in E' \end{cases}$$

By construction, this is a feasible matching: The edge constraints are obviously fulfilled on all $e \in E$. Since $f(\delta^-(v)) \leq kB$ for all $v \in V$, the edge constraints are also fulfilled for all $e \in E'$. Furthermore, the node constraints are fulfilled on s and on t. We now consider the nodes $v \in V' \setminus \{s,t\}$. Let $e_v \in E'$ be the edge connecting v^- and v^+ for some node $v \in V \setminus \{s,t\}$: By construction we have $m(\delta(v^-)) = m(\delta(v^-) \setminus \{e_v\}) + kB - f(\delta^-(v))$ and $m(\delta(v^-) \setminus \{e_v\}) = f(\delta^-(v))$. Hence, we obtain $m(\delta(v^-)) = kB$. Likewise, we have $m(\delta(v^+)) = m(\delta(v^+) \setminus \{e_v\}) + kB - f(\delta^-(v))$ and $m(\delta(v^+) \setminus \{e_v\}) = f(\delta^+(v))$ by construction. By assumption, f is a feasible flow and, since $v \notin \{s,t\}$, the flow conservation constraints are fulfilled. Hence, $f(\delta^-(v)) = f(\delta^+(v))$, which yields $m(\delta(v^+)) = kB$. This implies $m(\delta(v)) = kB$ for all $v \in V' \setminus \{s,t\}$.
Vice versa, let a feasible solution m to the instance I' of MWECBMMQ be given. For all edges $e \in E$ set $f_e := m_e$. By construction, all edge constraints are fulfilled. Let $v \in V \setminus \{s,t\}$ and $e_v = (v^-, v^+) \in E'$. We have $m(\delta(v^+) \setminus \{e_v\}) = f(\delta^+(v))$ and $m(\delta(v^-) \setminus \{e_v\}) = f(\delta^-(v))$ by construction. We show that the flow conservation constraints are fulfilled by proving that $m(\delta(v^+) \setminus \{e_v\}) = m(\delta(v^-) \setminus \{e_v\})$: If $m(\delta(v^+) \setminus \{e_v\}) = m(\delta(v^-) \setminus \{e_v\}) = 0$, the claim holds. We assume w.l.o.g. that $0 < m(\delta(v^+) \setminus \{e_v\})$. Since m is feasible, we have $m(\delta(v^+)) = kB$. Note

that by construction $m(\delta(v^+) \setminus \{e_v\}) \leq \sum_{e \in \delta(v^+) \setminus \{e_v\}} b_e = \sum_{e \in \delta^+(v)} b_e < kB$.
Hence, $0 < m(\delta(v^+) \setminus \{e_v\}) < kB$. In order for $m(\delta(v^+)) = kB$ to hold, we must have $m_{e_v} = kB - m(\delta(v^+) \setminus \{e_v\})$, which implies $0 < m_{e_v} < kB$. Then, in order for the minimum quantity constraint on v^- to be fulfilled, we must have $m(\delta(v^-)) = kB$, which implies $m(\delta(v^-) \setminus \{e_v\}) = kB - m_{e_v} = kB - (kB - m(\delta(v^+) \setminus \{e_v\})) = m(\delta(v^+) \setminus \{e_v\})$, which yields the claim. Hence, for MFMQ we have shown how a feasible solution to I can be transformed into a feasible solution to I' and vice versa.
Note that exactly the edges in $\delta(s)$ have weight 1 and all other edges have weight 0. Hence, the transformation maintains the objective value and a solution to I is optimal if and only if the corresponding solution to I' is optimal. This shows the claim regarding MFMQ.
We now consider MCFMQ: Let G denote the graph corresponding to I, the instance of MCFMQ, and let G' denote the graph corresponding to I', the instance of MWECBMMQ. In order to transform a solution to I into a solution to I' and vice versa, we use the same transformations that we have used for MFMQ. In addition, we set $m_{e_s} := 1$ if we transform a feasible solution to I into a solution to I'. We show that given an optimal solution m to I' we can either conclude infeasibility of I or derive an optimal solution to I.
In order to do so, we first show that there is a feasible solution to I' with $m_{e_s} = 1$ if and only if there is a feasible flow with flow value F in G:
Let a solution m to I' with $m_{e_s} = 1$ be given. Then we have $m(\delta(s) \setminus \{e_s\}) = F$. From our analysis of MFMQ we know that transforming the matching on the edges $E' \setminus \{e_s\}$ into a corresponding flow as described above yields a feasible s-t-flow with flow value F, i.e. a feasible solution to I. Thus, if there is a feasible solution m to I' for which $m_{e_s} = 1$, then there is a feasible solution to I.
Conversely, we have already seen for MFMQ how a feasible solution f to I can be turned into a matching m that is feasible for all nodes except for s and s' and where $m(\delta(s) \setminus \{e_s\}) = F$. Setting $m_{e_s} = 1$ as described above yields the claim.
Hence, we have shown for MCFMQ that there is a feasible solution to I' with $m_{e_s} = 1$ if and only if there is a feasible solution to I.
We now show, that if a feasible solution to I exists, then $m_{e_s} = 1$ in *every* optimal solution m to I' and m induces an optimal solution to I:
Let two feasible matchings m', m'' in G' be given. Then we have
$\sum_{e \in E \cup E' \setminus \{e_s\}} m'_e w_e - \sum_{e \in E \cup E' \setminus \{e_s\}} m''_e w_e = \sum_{e \in E \cup E' \setminus \{e_s\}} (m'_e - m''_e) w_e \leq \sum_{e \in E \cup E' \setminus \{e_s\}} \left| m'_e - m''_e \right| |w_e| \leq$
$B \sum_{e \in E} |c_e| < kB \left(\max_{e \in E} |c_e| + 1 \right) = w_{e_s}$. This implies that every feasible matching with $m_{e_s} = 1$ is better than every feasible matching with $m_{e_s} = 0$. As we have already seen, if there is a feasible solution to I, then there is a feasible solution to I' with $m_{e_s} = 1$. Hence, *every*

optimal solution to I' must have $m_{e_s} = 1$.
Let an optimal solution m to I' be given. Then $m_{e_s} = 1$, as we have just seen. We claim that applying the above transformation yields an *optimal* s-t-flow f in G. We have $\sum_{e \in E} f_e c_e = -\sum_{e \in E} m_e w_e = -\sum_{e \in E \cup E' \setminus \{e_s\}} m_e w_e$. Assume that f is not optimal, i.e. there is another s-t-flow f' for which $\sum_{e \in E} f'_e c_e < \sum_{e \in E} f_e c_e$. We create a feasible solution m' to I' by applying the above transformation to f'. In particular we set $m_{e_s} := 1$.
Then we have $-\sum_{e \in E \cup E' \setminus \{e_s\}} m'_e w_e = -\sum_{e \in E} m'_e w_e = \sum_{e \in E} f'_e c_e$. However, this implies $w_{e_s} + \sum_{e \in E \cup E' \setminus \{e_s\}} m'_e w_e > w_{e_s} + \sum_{e \in E \cup E' \setminus \{e_s\}} m_e w_e$, which contradicts optimality of m. Hence, f is also optimal. □

The previous theorem implies that MWECBMMQ is at least as hard as MFMQ and MCFMQ. On the other hand, we obtain the following corollary from the previous theorem:

Corollary 5.3.2. *Let a fixed integer $r \geq 1$ be given. MFMQ and MCFMQ can be solved in pseudo-polynomial time on graphs for which the treewidth is at most r.*

Proof. Note that if the treewidth of a given flow network G is bounded by r, then the treewidth of the graph G' constructed in the proof of Theorem 5.3.1 is bounded by $2r$: We obtain a tree-decomposition of G' from a tree-decomposition of G by replacing every node v by the respective v^- and v^+ (or s by s and s') in all X_i. Hence, the claim follows from Theorem 5.3.1 and Theorem 4.3.12. □

As we have already mentioned, the authors of [47] show that MFMQ can be solved on series parallel graphs in polynomial time if the minimum quantities are uniform. We extend this result to the case that the number of different minimum quantities is bounded by a fixed integer r:

Corollary 5.3.3. *For a fixed integer r, MFMQ can be solved in polynomial time on series parallel graphs if the number of different minimum quantities is bounded by r.*

Proof. We show that the complexity of the dynamic program given in [47] (with a small adaptation) remains polynomial.
Let $G = (V, E)$ denote the series parallel graph corresponding to the instance of MFMQ and set $m := |E|$. W.l.o.g. we assume that there are exactly r different minimum quantities. Else, we add dummy minimum quantities that do not occur in the graph. Let $\{q_1, \ldots, q_r\}$ be the set of different minimum quantities.
The dynamic program provided by the authors of [47] successively computes sets $S_{G'}(k)$ for series parallel subgraphs G' of G along a decomposition of G. These sets contain the values

of feasible s-t-flows in G' for which the flow is positive on exactly k edges (i.e. $f_{e_i} \in \{q_{e_i}, b_{e_i}\}$ for the respective edges e_i).
The dynamic program works as follows:

- $G' = K_2$
  Let $q_e \in \{q_1, \dots, q_r\}$ and b_e denote the minimum quantity and the capacity on the single edge e of G'. The only change we make to the dynamic program given in [47], is using the *edge-specific* minimum quantity q_e in the following computation (instead of the *uniform* minimum quantity):

$$S_{G'}(k) \coloneqq \begin{cases} \{0\} & \text{if } k = 0 \\ [q_e, b_e] & \text{if } k = 1 \\ \emptyset & \text{if } k > 1 \end{cases}$$

- Series composition
  The series composition is the same as the one provided in [47]. Let G' be the series composition of G_1 and G_2. Then $S_{G'}(k)$ can be computed as follows:
  $S_{G'}(k) \coloneqq \bigcup_{\substack{0 \le k_1, k_2 \le k \\ k_1 + k_2 = k}} S_{G_1}(k_1) \cap S_{G_2}(k_2)$

- Parallel composition
  The parallel composition is also the same as the one in [47]. Let G' be the parallel composition of G_1 and G_2. Then $S_{G'}(k)$ can be computed as follows, where "+" denotes the element-wise sum (Minkowski sum):
  $S_{G'}(k) \coloneqq \bigcup_{\substack{0 \le k_1, k_2 \le k \\ k_1 + k_2 = k}} S_{G_1}(k_1) + S_{G_2}(k_2)$

Once the sets $S_G(k)$ have been determined for all $k \in \{1, \dots, m\}$, the maximum flow of the instance of MFMQ is given by $\max_{k \in \{1,\dots,m\}} \max S_G(k)$.
As in [47], correctness of the algorithm follows by construction. We still have to show that the complexity of the algorithm is polynomial. The argumentation works analogously to [47], but in addition, we take the different structure of the sets $S_{G'}(k)$ into account and show that the complexity of the algorithm remains in fact polynomial. Set $B \coloneqq \sum_{e \in E} b_e$. Let $p_{G'}(j)$ denote the number of edges in some subgraph G' of G that have minimum quantity q_j. Set $I_{G'} \coloneqq \left\{ \sum_{j=1}^{r} c_j q_j \,\middle|\, 0 \le c_j \le p_{G'}(j) \text{ for all } j = 1, \dots, r \right\}$. Note that $p_{G'}(j) \le m$ for all j. Hence, we have that $|I_{G'}| \le (m+1)^r$. Furthermore, note that for all graphs G' the elements of $I_{G'}$ are

linear combinations of the different minimum quantities that occur in G. The sets $I_{G'}$ only differ in the ranges of the coefficients c_j.

We claim that for every G' all $S_{G'}(k)$ can be expressed as $S_{G'}(k) = \bigcup_{i \in I_{G'}(k)} [i, U_{G'}(k,i)]$ for some $I_{G'}(k) \subseteq I$ and some $U_{G'}(k,i) \in [i, \dots, B]$. Note that the intervals are not required to be disjoint.

If $G' = K_2$ where e is the edge in G', then the claim holds: If $k = 0$, set $I_{G'}(0) \coloneqq \{0\} \subseteq I$ and $U_{G'}(0,0) = 0$. If $k = 1$, $I_{G'}(1) \coloneqq \{q_e\} \subseteq I$ and set $U_{G'}(1, q_e) = b_e$. If $k > 1$, set $I_{G'}(k) \coloneqq \emptyset \subseteq I$.

If G' is the series composition of G_1 and G_2 and the statement holds for G_1 and G_2, then it is obvious that the intersection maintains this property and the statement also holds for G'.

Let G' be the parallel composition of G_1 and G_2 and let the statement be true for G_1 and G_2. We have that $p_{G'}(j) = p_{G_1}(j) + p_{G_2}(j)$ for all $j \in \{1, \dots, r\}$. Hence, if $i_1 \in I_{G_1}$ and $i_2 \in I_{G_2}$, then $i_1 + i_2 \in I_{G'}$. Furthermore, the sum of the maximum flow through G_1 and the maximum flow through G_2 can never exceed B, which implies $\max_{k \in \{1,\dots,m\}} \max S_{G_1}(k) + \max_{k \in \{1,\dots,m\}} \max S_{G_2}(k) \leq B$. Hence, given arbitrary $0 \leq k_1, k_2, k \leq m$ for which $k = k_1 + k_2$ and arbitrary intervals $[i_1, U_{G_1}(k_1, i_1)] \subseteq S_{G_1}(k_1)$ and $[i_2, U_{G_2}(k_2, i_2)] \subseteq S_{G_2}(k_2)$, we have that $[i_1 + i_2, U_{G_1}(k_1, i_1) + U_{G_2}(k_2, i_2)]$ can be represented as claimed. Since $S_{G'}(k)$ is the Minkowski sum of $S_{G_1}(k_1)$ and $S_{G_2}(k_2)$, the statement holds for G'.

The claim now follows by induction. This implies that every $S_{G'}(k)$ can be represented as the union of at most $(m+1)^r$ intervals.

We now consider the number of steps required to compute $S_{G'}(k)$ for all $k \in \{1, \dots, m\}$ for a given G': If $G' = K_2$, then $S_{G'}(k)$ can be computed in constant time for k fixed. Computing $S_{G'}(k)$ for all $k \in \{1, \dots, m\}$ requires time $O(m)$. Now assume that G' is the parallel or series composition of G_1 or G_2 and that $S_{G_1}(k_1)$ and $S_{G_2}(k_2)$ have been computed for all $k_1, k_2 \in \{1, \dots, m\}$. Then for k_1, k_2 fixed, $S_{G_1}(k_1) \cap S_{G_2}(k_2)$ or $S_{G_1}(k_1) + S_{G_2}(k_2)$, respectively, can be computed in $O(m^{2r})$: The intersection and the Minkowski sum process each pair of intervals from $S_{G_1}(k_1)$ and $S_{G_2}(k_2)$. Each of the sets $S_{G_1}(k_1)$ and $S_{G_2}(k_2)$ consists of at most $(m+1)^r$ intervals and for processing a pair of intervals only the endpoints of the intervals are required. In particular, the length of the intervals does not matter in terms of complexity, as long as the endpoints of the intervals remain polynomially bounded in the size of the input instance, which holds by construction. Hence, considering all $0 \leq k_1, k_2 \leq k$ for which $k_1 + k_2 = k$ for k fixed requires $O(km^{2r})$ steps and computing $S_{G'}(k)$ for all $k \in \{1, \dots, m\}$ requires at most $O(m^{2r+2})$ steps. The dynamic program terminates after at most m parallel or series compositions. Thus, it finishes after at most $O(m^{2r+3} + m) = O(m^{2r+3})$ steps (including initializing the dynamic program for all $G' = K_2$). Finally, $\max_{k \in \{1,\dots,m\}} \max S_G(k)$ has to be computed. For k fixed, determining $\max S_G(k)$ requires at most $O(m^r)$ steps for comparing

$U_G(k,i)$ for all $i \in I_G(k)$. Considering all $k \in \{1,\dots,m\}$ increases the number of steps required to $O(m^{r+1})$. We assume that a decomposition (i.e. an sp-tree) of G has already been computed, else it can be computed in polynomial time (Lemma 1.4.2). Hence, the overall complexity of determining an optimal solution is $O(m^{2r+3} + m + m^{r+1}) = O(m^{2r+3})$, which is in fact polynomial due to r being fixed by assumption. □

We have shown that there are polynomial-time bicriteria (α,β)-approximation algorithms for all classes of problems that we have considered in this thesis so far. We now show that unless P=NP, this is not possible for MCFMQ and MCRFMQ, using a similar reduction as the one given in [32]. If we consider flow problems, the approximation factor β refers to the minimum quantity and capacity constraints defined on the edges of the graph.

Theorem 5.3.4. *MCFMQ and MCRFMQ are NP-hard and unless P=NP, there cannot be any polynomial-time bicriteria approximation algorithm for these problems, even on series parallel or on bipartite graphs.*

Proof. Let an instance of PARTITION be given by n sizes $s_i \in \mathbb{N}$ $(i = 1,\dots,n)$ where $\sum_{i=1}^{n} s_i = B$. Create an instance of MCFMQ or MCRFMQ as follows: Set $V := \{s,t\}$, $E := \{e_1,\dots,e_n\}$ where $e_i = (s,t)$ for all $i \in \{1,\dots,n\}$, $q_{e_i} := b_{e_i} := s_i$ for all $i \in \{1,\dots,n\}$, $c_{e_i} := 0$ for all $i \in \{1,\dots,n\}$ and $F := \frac{B}{2}$.
It is obvious that there is a feasible solution with flow value F if and only if a feasible solution to the given instance of PARTITION exists. By Definition 1.3.9, a bicriteria approximation algorithm returns an approximate solution if there is a feasible solution to the given instance, else it returns infeasibility of the instance. Hence, a bicriteria approximation algorithm for MCFMQ or MCRFMQ would solve PARTITION. This yields the claim. □

Note that different definitions of bicriteria approximation algorithms exist that might lead to different results (Remark 1.3.10).
Even though a polynomial-time bicriteria approximation algorithm for MCRFMQ does not exists, such an algorithm exists for MRFMQ on series parallel graphs. In order to show this claim, we require the following lemma:

Lemma 5.3.5. *For every instance of MRFMQ there is an integral optimal solution.*

Proof. Let an instance I of MRFMQ and an arbitrary optimal flow f be given. If f has flow value 0, a zero-flow with $f_e = 0$ for all edges e is an integral optimal solution to I. Else, we remove all edges from the graph corresponding to I for which $f_e = 0$. Let E' denote the set of remaining edges. We interpret the minimum quantities on the remaining edges as *fixed* lower bounds, i.e. we require $f_e \geq q_e$ for all $e \in E'$. We denote this instance by I'. Note that

f restricted to the edges in E' is an optimal solution to I' (Otherwise, this would contradict optimality of f in I). By definition of MRFMQ, all lower bounds and edge capacities are integral. Then there is an integral solution f' to I' that has the same flow value as f: This follows either from the fact that the problem can be formulated as an integer program with a totally unimodular matrix or from the results regarding the maximum flow problem with lower bounds given in [1]. Obviously, f' is also a feasible solution to I. □

We now show the existence of a bicriteria approximation algorithm for MRFMQ on series parallel graphs. The outline of the algorithm and the proof are similar to Theorem 4.3.18 but several adjustments are included.

Theorem 5.3.6. *For $\epsilon > 0$ fixed, there is a polynomial $(1,\ 1+\epsilon)$-approximation algorithm for MRFMQ on series parallel graphs.*

Proof. We provide a dynamic program that finds an approximate solution as described in the claim. As in the proof of Theorem 4.3.18, we assume that an arbitrary but fixed way of constructing the graph (i.e. an sp-tree) is given. Again, in order to achieve a polynomial complexity of the algorithm, we only consider a subset of all feasible flows during each construction step of the series parallel graph. Each partial solution is identified by the flow value through the respective graph G', where G' denotes the subgraph of G constructed in the current iteration. The flow values to be considered by the algorithm are characterized by the set J. For each value $a \in J$, the algorithm determines whether there is an s-t-flow with flow value a through G' that fulfills certain conditions. If so, one such flow is chosen to be stored. In addition, there is a function $W_{G'} : J \to \{0, 1\}$ that indicates for each value in $a \in J$ whether the algorithm has stored an intermediate solution with flow value a. The crucial step of the algorithm is the cleansing step that is carried out after each parallel composition. The cleansing step ensures that the complexity of the algorithm remains polynomial.
Let an instance I of MRFMQ be given on a graph $G = (V, E)$.
Set

- $f_{\max} := \sum_{e \in E} b_e$
- $\sigma := \sqrt[m]{1+\epsilon} - 1$ where $m = |E|$
- $k := \lceil \log_{1+\sigma} f_{\max} \rceil + 1 = \lceil m \log_{1+\epsilon} f_{\max} \rceil + 1$

This implies $k \in O(m \log \sum_{e \in E} b_e)$, so it is polynomial in the encoding size of I.

In addition, we define the following intervals and values:

- $I_0 := \{0\}$
- $I_1 := \left[(1+\sigma)^0, (1+\sigma)^1\right] = [1, (1+\sigma)]$
- $I_i := \left((1+\sigma)^{i-1}, (1+\sigma)^i\right]$ for all $2 \leq i \leq k$
- $p_0 := 0$
- $p_i := (1+\sigma)^i$ for all $1 \leq i \leq k$

This allows us to denote the intervals I_i using the values p_i:

- $I_0 = \{p_0\} = \{0\}$
- $I_1 = [1, p_1]$
- $I_i = (p_{i-1}, p_i]$ for all $2 \leq i \leq k$

We make the following observations regarding the above definitions:

- We have $\{0\} \cup [1, f_{\max}] \subseteq \bigcup_{i=0}^{k-1} I_i$. Note that $f_{\max}$ is an upper bound on the optimal flow value.
- For all $1 \leq i \leq k$, we have $\frac{\max I_i}{\inf I_i} \leq 1+\sigma$.

We now describe how the algorithm works for each of the three possible situations, which are the basic one-edge graph K_2, the series composition and the parallel composition. We assume that the graph G' with terminal nodes s and t is the graph to be constructed in the current iteration. We use the same index for each series parallel graph and its respective source and sink, i.e. the terminals of graph G_i are denoted by s_i and t_i.
We set $J := \{p_i \mid i = 0, \ldots, k-1\}$ and $W_{G'}(p_0) := 1$ for all subgraphs G' of G that are considered by the algorithm. In addition, we set $J' := J + J$ (Minkowski sum). We associate the zero-flow with the flow value $p_0 = 0$ for all graphs G'.

- $G' = K_2$
  We denote the (only) edge of the graph by $e = (s,t)$ and define $W_{G'}$ for all $\{p_i \mid i = 1, \ldots, k-1\}$ as follows:

$$W_{G'}(p_i) := \begin{cases} 1 & \text{if } p_i \in [q_e, (1+\sigma)b_e] \\ 0 & \text{else} \end{cases}$$

- Series composition
  Let G' be constructed from G_1 and G_2 where t_1 and s_2 are identified.
  We assume that W_{G_1} and W_{G_2} have been determined in a previous iteration. For each $i \in \{1,\dots,k-1\}$ set $W_{G'}(p_i) \coloneqq \min\{W_{G_1}(p_i), W_{G_2}(p_i)\}$. If $W_{G'}(p_i) = 1$, store the union of the (edge-disjoint) partial flows that have been stored for p_i in G_1 and G_2 as the corresponding flow in G'.

- Parallel composition
  Let G' be constructed from G_1 and G_2 where s_1 and s_2 as well as t_1 and t_2 are identified.
  Assume again that W_{G_1} and W_{G_2} have been determined in an earlier iteration. For each element $a \in J'$, $W'_{G'}$ is defined as follows:
  $W'_{G'}(a) \coloneqq \max\{\min\{W_{G_1}(p_{i_1}), W_{G_2}(p_{i_2})\} \mid p_{i_1}, p_{i_2} \in J \wedge a = p_{i_1} + p_{i_2}\}$
  By construction, for each value $a \in J'$ there is at least one pair of values $p_{i_1}, p_{i_2} \in J$ so that $a = p_{i_1} + p_{i_2}$. Hence, the set over which the maximum is determined is non-empty and the above value is well-defined. In order to reduce the number of values stored, we omit $W'_{G'}$ after the iteration and only store $W_{G'}$ (which is defined on J instead of J'). Recall that we have already set $W_{G'}(p_0) = 1$. For all $i \in \{1,\dots,k-1\}$ the function $W_{G'}$ is defined as follows:
  $W_{G'}(p_i) \coloneqq \max\left\{W'_{G'}(a) \,\middle|\, a \in J' \cap (I_i \cup I_{i+1})\right\}$
  If $W_{G'}(p_i) = 1$ for some p_i, then by construction there are $p_{i_1}, p_{i_2} \in J$ for which $W_{G_1}(p_{i_1}) = W_{G_2}(p_{i_2}) = 1$ and $a = p_{i_1} + p_{i_2}$. If there are multiple such tuples (p_{i_1}, p_{i_2}), an arbitrary tuple is selected. Note that $J \subseteq J'$ and $J \cap I_i \neq \emptyset$ for all $i \in \{1,\dots,k-1\}$. Hence, $W_{G'}(p_i)$ is well-defined. We scale the edge flows of the (edge-disjoint) partial solutions that are associated with p_{i_1} in G_1 and with p_{i_2} in G_2 by a factor $\frac{p_i}{a}$ and associate the resulting flow through G' with $p_i \in J$. Note that this step scales the flow up or down by a factor at most $1 + \sigma$.

Once a series or a parallel composition is considered by the algorithm, we require W_{G_1} and W_{G_2} to be defined for all values p_i in order for all steps of the algorithm to be well-defined. This is in fact the case: For $G' = K_2$ we have that $W_{G'}$ is defined for all values p_i by construction. If G' is the series or the parallel composition of G_1 and G_2 and W_{G_1} and W_{G_2} are defined for all values p_i, then so is $W_{G'}$. Hence, the claim follows by induction.
The dynamic program continues until $G' = G$. The approximate objective value that we denote by APP is given by $\text{APP} \coloneqq \max\left\{p_i \in J \,\middle|\, W_G(p_i) = 1\right\}$.
We introduce the following notation for an arbitrary subgraph G' that occurs during the series parallel composition of G:

- Given an arbitrary but fixed optimal solution to the given instance of MRFMQ, $f_{G'}^{\text{OPT}}$ denotes the flow value through G' and $f_{G',e}^{\text{OPT}}$ denotes the flow on $e \in E(G')$ in the given solution. According to Lemma 5.3.5 all edge flows can be assumed to be integral.
- For some $p_i \in J$, $f_{G',p_i,e}^{\text{ALG}}$ denotes the flow on $e \in E(G')$ corresponding to the partial solution returned by the algorithm with flow value p_i in G'.

We claim that for all subgraphs G' of G considered by the algorithm the following holds:

1. If $f_{G'}^{\text{OPT}} \in I_i$ for some $i \in \{0,\dots,k-1\}$, then $W_{G'}(p_i) = 1$.
2. If $W_{G'}(p_i) = 1$ for some $p_i \in J$, then $f_{G',p_i,e}^{\text{ALG}} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G')|}}, (1+\sigma)^{|E(G')|} b_e\right]$ for all $e \in E(G')$.

Note that both parts of the claim hold for $i = 0$ for all subgraphs G' of G, since $W_{G'}(p_0) = 1$ and the zero-flow on the respective graph is associated with p_0 by construction. Hence we only have to show the claim for $i \in \{1,\dots,k-1\}$. We show the claim by induction:
The algorithm always starts with some graph $G' = K_2$, which is the base case of our induction.
We show the first part of the claim:
We have $|E(G')| = 1$. Let e denote the single edge of G'. We have $q_e \le f_{G',e}^{\text{OPT}} = f_{G'}^{\text{OPT}} \le b_e$ and $q_e \le f_{G',e}^{\text{OPT}} \le p_i \le (1+\sigma) f_{G',e}^{\text{OPT}} \le (1+\sigma) b_e$. This implies $W_{G'}(p_i) = 1$ and yields the first part of the claim.
We now assume $W_{G'}(p_i) = 1$. Recall that $i \ge 1$. By construction, this implies $f_{G',p_i,e}^{\text{ALG}} = p_i \in [q_e, (1+\sigma) b_e]$. Hence, the second part of the claim is also fulfilled.

For the induction step we consider the series composition first:
We show the first part of the claim: Let G' be constructed from G_1 and G_2 and $f_{G'}^{\text{OPT}} = f_{G_1}^{\text{OPT}} = f_{G_2}^{\text{OPT}} \in I_i$ for some $i \in \{1,\dots,k-1\}$. We assume that the claim holds for p_i in G_1 and G_2, i.e. $W_{G_1}(p_i) = W_{G_2}(p_i) = 1$. Hence, $W_{G'}(p_i) = \min\{W_{G_1}(p_i), W_{G_2}(p_i)\} = 1$. This yields the first part of the claim.
For proving the second part of the claim, we assume that $W_{G'}(p_i) = 1$ for some $i \ge 1$. Then, by construction, $W_{G_1}(p_i) = W_{G_2}(p_i) = 1$. Hence, the assumption holds for the edge flows associated with p_i in G_1 and in G_2:

$$f_{G_1,p_i,e}^{\text{ALG}} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_1)|}}, (1+\sigma)^{|E(G_1)|} b_e\right] \text{ for all } e \in E(G_1)$$

and analogously

$$f_{G_2,p_i,e}^{\text{ALG}} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_2)|}}, (1+\sigma)^{|E(G_2)|} b_e\right] \text{ for all } e \in E(G_2).$$

Since $E(G') = E(G_1) \dot\cup E(G_2)$, this implies

$$f^{\mathrm{ALG}}_{G',p_i,e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G')|}}, (1+\sigma)^{|E(G')|} b_e \right] \text{ for all } e \in E(G').$$

Thus, we have established the second part of the claim.

We now consider the parallel composition and we begin by showing the first part of the claim: Let G' be constructed from G_1 and G_2 and $f^{\mathrm{OPT}}_{G'} \in I_i$ for some $i \in \{1,\dots,k-1\}$. Then $f^{\mathrm{OPT}}_{G'} = f^{\mathrm{OPT}}_{G_1} + f^{\mathrm{OPT}}_{G_2}$ where $f^{\mathrm{OPT}}_{G_1} \in I_{i_1}$ and $f^{\mathrm{OPT}}_{G_2} \in I_{i_2}$ for some $i_1, i_2 \in \{0,\dots,k-1\}$. According to the induction hypothesis the claim holds regarding $f^{\mathrm{OPT}}_{G_1}$ in G_1 and $f^{\mathrm{OPT}}_{G_2}$ in G_2, respectively, i.e. $W_{G_1}(p_{i_1}) = W_{G_2}(p_{i_2}) = 1$. Since $p_{i_1} \in I_{i_1}$, $f^{\mathrm{OPT}}_{G_1} \in I_{i_1}$ and $p_{i_1} \geq f^{\mathrm{OPT}}_{G_1}$, we have $f^{\mathrm{OPT}}_{G_1} \leq p_{i_1} \leq (1+\sigma) f^{\mathrm{OPT}}_{G_1}$. Analogously, we have $f^{\mathrm{OPT}}_{G_2} \leq p_{i_2} \leq (1+\sigma) f^{\mathrm{OPT}}_{G_2}$. This implies $f^{\mathrm{OPT}}_{G'} \leq p_{i_1} + p_{i_2} \leq (1+\sigma) f^{\mathrm{OPT}}_{G'}$. Hence, if $f^{\mathrm{OPT}}_{G'} \in I_i$, then $p_{i_1} + p_{i_2} \in I_i \cup I_{i+1}$. Note that we have $i \in \{1,\dots,k-1\}$, which implies that I_{i+1} is well-defined. By construction, $p_{i_1} + p_{i_2} \in J'$ and $W'_{G'}(p_{i_1} + p_{i_2}) = 1$. Thus, we have $W_{G'}(p_i) = \max\left\{ W'_{G'}(a) \,\middle|\, a \in J' \cap (I_i \cup I_{i+1}) \right\} = W'_{G'}(p_{i_1} + p_{i_2}) = 1$.

We now show the second part of the claim. Assume that $W_{G'}(p_i) = 1$ for some $i \in \{1,\dots,k-1\}$. Then there is some value $a \in J' \cap (I_i \cup I_{i+1})$ for which $W'_{G'}(a) = 1$. We have $a = p_{i_1} + p_{i_2}$ for some $p_{i_1}, p_{i_2} \in J$ with $W_{G_1}(p_{i_1}) = W_{G_2}(p_{i_2}) = 1$. By assumption, the claim holds for the partial solutions associated with $p_{i_1}, p_{i_2} \in J$:

$$f^{\mathrm{ALG}}_{G_1,p_{i_1},e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_1)|}}, (1+\sigma)^{|E(G_1)|} b_e \right] \text{ for all } e \in E(G_1)$$

and analogously

$$f^{\mathrm{ALG}}_{G_2,p_{i_2},e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_2)|}}, (1+\sigma)^{|E(G_2)|} b_e \right] \text{ for all } e \in E(G_2).$$

The flow in G' associated with p_i is computed by the algorithm by scaling the edge flows of the solutions associated with flow value p_{i_1} in G_1 and p_{i_2} in G_2 up or down to p_i, i.e. by a factor at most $1+\sigma$. Hence, we obtain

$$f^{\mathrm{ALG}}_{G',p_i,e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_1)|+1}}, (1+\sigma)^{|E(G_1)|+1} b_e \right] \text{ for all } e \in E(G_1)$$

and analogously

$$f^{\mathrm{ALG}}_{G',p_i,e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G_2)|+1}}, (1+\sigma)^{|E(G_2)|+1} b_e \right] \text{ for all } e \in E(G_2).$$

We have that $E(G') = E(G_1) \dot\cup E(G_2)$, $|E(G_1)| \geq 1$ and $|E(G_2)| \geq 1$, which implies $|E(G')| \geq |E(G_1)| + 1$ and $|E(G')| \geq |E(G_2)| + 1$, which implies

$$f^{\text{ALG}}_{G',p_i,e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G')|}}, (1+\sigma)^{|E(G')|} b_e \right] \text{ for all } e \in E(G').$$

Thus, we have established the second part of the claim. This completes the induction.
Once the algorithm has computed W_G, the approximate solution is determined. We have $f^{\text{OPT}}_G \in I_i$ for some $i \in \{0, \ldots, k-1\}$. Hence, from the previous induction we get that $\text{APP} = \max\left\{ p_i \in J \,\middle|\, W_G(p_i) = 1 \right\} \geq p_i \geq f^{\text{OPT}}_G$. Since $W_G(\text{APP}) = 1$, we get that $f^{\text{ALG}}_{G,\text{APP},e} \in \{0\} \cup \left[\frac{q_e}{1+\sigma^{|E(G)|}}, (1+\sigma)^{|E(G)|} b_e \right]$ for all $e \in E(G)$. By construction, $(1+\sigma)^{|E(G)|} = (1+\sigma)^m = 1+\epsilon$. Thus, we get $f^{\text{ALG}}_{G,\text{APP},e} \in \{0\} \cup \left[\frac{q_e}{1+\epsilon}, (1+\epsilon) b_e \right]$ for all $e \in E(G)$.
We have shown that the algorithm determines a flow through G for which the flow value is at least the optimal flow value. By construction, the approximate solution violates the minimum quantity constraints and the capacity constraints by a factor at most $1+\epsilon$. We still have to show that the flow conservation constraints are fulfilled, i.e. $\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = 0$ for all $v \in V \setminus \{s,t\}$. This follows by induction: For the base case $G = K_2$ the claim holds, since $V \setminus \{s,t\} = \emptyset$. For the series and the parallel composition, let G' be the composition of G_1 and G_2 and let the claim hold for all $v \in V(G_1) \setminus \{s_1, t_1\}$ and for all $v \in V(G_2) \setminus \{s_2, t_2\}$. The parallel composition does not change the set of non-terminal nodes. Hence, the claim holds for the parallel composition and scaling all edge flows by the same factor does not affect the flow conservation constraints. The series composition by the algorithm does not affect the flow conservation constraints regarding the non-terminal nodes in G_1 and G_2. However, the terminal nodes t_1 and s_2 are merged and become a non-terminal node in G'. By construction, the union of partial solutions from G_1 and G_2 only takes place if the outflow of t_1 and the inflow of s_2 are both p_i for some $p_i \in J$ and such partial solutions exist. Hence, the flow conservation constraint is also fulfilled for the non-terminal node t_1 $(= s_2)$ in G'. This completes the induction and shows that the flow conservation constraints are fulfilled for all non-terminal nodes in G.
Note that according to our definition, an (α, β)-approximation has to return infeasibility of the given instance if there is no feasible solution to the instance. However, since a zero-flow is a feasible solution to every instance of MRFMQ, infeasibility can never occur.
Hence, we have shown that the above algorithm is in fact a $(1, 1+\epsilon)$-approximation.
We now consider the complexity of the algorithm. The complexity analysis is similar to the one in Theorem 4.3.18:
As an input we assume a series parallel graph G as well as a sequence of series and parallel compositions for constructing G. Given a series parallel graph, such a sequence can be

computed in polynomial time and the number of subgraphs of G that occur during this sequence of compositions is bounded by $O(|E|)$ (Lemma 1.4.2).
By construction $J = \{p_i \mid i = 0, \ldots, k-1\}$, $J' = J + J$ and $W_{G'}(p_0) = 1$ for all subgraphs G' of G that are considered by the algorithm. Hence, initializing J, J' and $W_{G'}$ for all G' requires at most $O(|E| + k^2)$ steps.
Once a sequence of compositions has been determined, the above algorithm can be applied. We first consider the one-edge graphs K_2: There are $|E|$ such graphs and $k-1$ values are computed in constant time for each of them. Hence, processing all one-edge graphs requires a number of steps bounded by $O(|E|k)$.
Once all one-edge graphs have been considered by the algorithm, the graph G can be constructed iteratively by series and parallel compositions. As we have already noted, the number of these iterations is bounded by $O(|E|)$:
For each iteration in which a series composition is considered we have to compute $W_{G'}$ for $k-1$ values (since $W_{G'}(p_0)$ has already been initialized) and each computation can be processed in constant time. Hence, the computational complexity of each such iteration is in $O(k)$.
For each parallel composition the algorithm computes $W'_{G'}(a)$ for a specific $a \in J'$ in a number of steps bounded by $O(k^2)$. Hence, determining $W'_{G'}(a)$ for all $a \in J'$ requires a number of steps bounded by $O(k^4)$. Once all values for $W'_{G'}$ have been computed, $W_{G'}$ is computed for $k-1$ values. Note that every element of J' is considered at most twice during the computation of $W_{G'}$: If $a \in J' \cap I_{i+1}$ is considered during the computation of $W_{G'}(p_i)$, then it is also considered when $W_{G'}(p_{i+1})$ is determined (unless $i = k-1$). Hence, the number of steps required for determining the value of $W_{G'}$ for all p_i is bounded from above by $O(k^4)$.
For each p_i a partial flow is stored, which might require rounding the edge flows. Hence, storing the partial solutions requires $O(|E|k)$ steps. Thus, the complexity of each iteration that processes a parallel composition is bounded from above by $O(|E|k^4)$.
After all iterations have been completed, APP is determined by computing the maximum over a set with k values, which implies that the complexity of this step is in $O(k)$.
All in all, we obtain the computational complexity $O(|E|^2 k^4)$. As k is polynomial in the encoding size of the instance, so is the complexity of the algorithm.

□

We now consider the *generalized* flow problems GMFMQ and GMCFMQ and show how they can be transformed into the *non-generalized* flow problems MFMQ and MCFMQ.
Techniques for transforming generalized flow problems *without* minimum quantity constraints into non-generalized flow problems *without* minimum quantity constraints have already been considered by other authors: In particular, the authors of [17] present a scaling technique

that transforms generalized flow problems that have an incidence matrix that does not have full row rank into non-generalized flow problems.

Theorem 5.3.7. *An instance I of GMFMQ can be transformed into an equivalent instance I' of MFMQ in polynomial time. Likewise, GMCFMQ can be transformed into an equivalent instance of MCFMQ in polynomial time.*

Proof. Let an instance I of GMFMQ or GMCFMQ be given. Let $G = (V, E)$ denote the respective graph. We show how the instance of GMFMQ or GMCFMQ can be transformed into an equivalent instance of MFMQ or MCFMQ. The transformation replaces every edge $e \in E$ with flow multiplier μ_e by a series parallel graph *without* flow multipliers. In order to ensure that the outflow of the respective series parallel subgraph is exactly μ_e times the inflow, special edges are added that allow additional inflow into or outflow out of the series parallel subgraph. At this point minimum quantity constraints are essential to ensure that the inflow and outflow on these special edges is exactly the amount of flow that would be added or removed by the multiplier μ_e in G. Hence, the transformation would not work without minimum quantity constraints.
We now describe the steps that have to be carried out for every edge of G. Let an arbitrary edge $e = (v_1, v_2) \in E$ as shown in Figure 5.3.4 be given. Let $\mu_e = \frac{y_e}{z_e}$. We assume that all fractional edge multipliers are given in simplest form so that y_e and z_e are relatively prime.

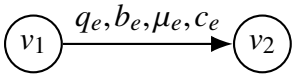

Fig. 5.3.4 Input edge GMFMQ / GMCFMQ; The edge labels represent the minimum quantities, capacities, multipliers and costs

For each edge e we apply the following steps. Unless specified otherwise, we consider an input instance of GMCFMQ. The same reasoning holds for GMFMQ (except for the edge costs and the flow value constraint $\sum_{e \in \delta^+(s)} f_e = F$).

1. Remove the edge e from the graph.
2. Set $k_{e,1} := \lfloor \log b_e \rfloor$.
3. If $z_e \geq 2$, set $k_{e,2} := \lfloor \log(z_e - 1) \rfloor$.
4. If $z_e \geq 2$, set $k_{e,3} := \left\lfloor \log_{z_e}(y_e b_e) \right\rfloor$.
5. Create nodes $a_{e,1}$, $a_{e,2}$ and $a_{e,3}$.

6. Create nodes g_{e,l_1} for all $0 \leq l_1 \leq k_{e,1}$.
7. If $z_e \geq 2$, create nodes h_{e,l_2,l_3} for all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$.
8. Define the demand of the above nodes to be 0.
9. Create an edge $e' = (v_1, a_{e,1})$ with $q_{e'} := q_e$, $b_{e'} := b_e$ and (if applicable) $c_{e'} := c_e$.
10. Create edges $e' = (a_{e,1}, g_{e,l_1})$ for all $0 \leq l_1 \leq k_{e,1}$ with $q_{e'} := b_{e'} = 2^{l_1}$ and $c_{e'} = 0$.
11. Create edges $e' = (t, g_{e,l_1})$ for all $0 \leq l_1 \leq k_{e,1}$ with $q_{e'} := b_{e'} := 2^{l_1}(y_e - 1)$ and $c_{e'} = 0$.
12. Create edges $e' = (g_{e,l_1}, a_{e,2})$ for all $0 \leq l_1 \leq k_{e,1}$ with $q_{e'} := b_{e'} := 2^{l_1} y_e$ and $c_{e'} = 0$.
13. If $z_e = 1$, create an edge $e' = (a_{e,2}, a_{e,3})$ with $q_{e'} := 0$, $b_{e'} := y_e b_e$ and $c_{e'} = 0$.
14. If $z_e \geq 2$, create edges $e' = (a_{e,2}, h_{e,l_2,l_3})$ for all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$ with $q_{e'} := b_{e'} := 2^{l_2} z_e^{l_3}$ and $c_{e'} = 0$.
15. If $z_e \geq 2$, create edges $e' = (h_{e,l_2,l_3}, t)$ for all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$ with $q_{e'} := b_{e'} := 2^{l_2} z_e^{l_3 - 1}(z_e - 1)$ and $c_{e'} = 0$.
16. If $z_e \geq 2$, create edges $e' = (h_{e,l_2,l_3}, a_{e,3})$ for all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$ with $q_{e'} := b_{e'} := 2^{l_2} z_e^{l_3 - 1}$ and $c_{e'} = 0$.
17. Create an edge $e' = (a_{e,3}, v_2)$ with $q_{e'} := 0$, $b_{e'} := y_e b_e$ and $c_{e'} = 0$.
18. If I is an instance of MCFMQ and F is the required flow value, then the same flow value is required in I'.

In addition, we create a node t' and an edge $e = (t, t')$ with $q_e := 0$, $b_e := \sum_{e' \in \delta_G^+(s)} b_{e'}$ and $c_e := 0$. We defined the demand of t to be 0.

The following figures show the output for the above edge as well as the nodes s, t, and t' and the edges incident to those nodes, assuming $k_{e,1} = 3$, $k_{e,2} = 2$ and $k_{e,3} = 2$. The first figure shows the case $z_e = 1$, the second figure shows the case $z_e \geq 2$. For the sake of clarity, most edge labels are omitted.

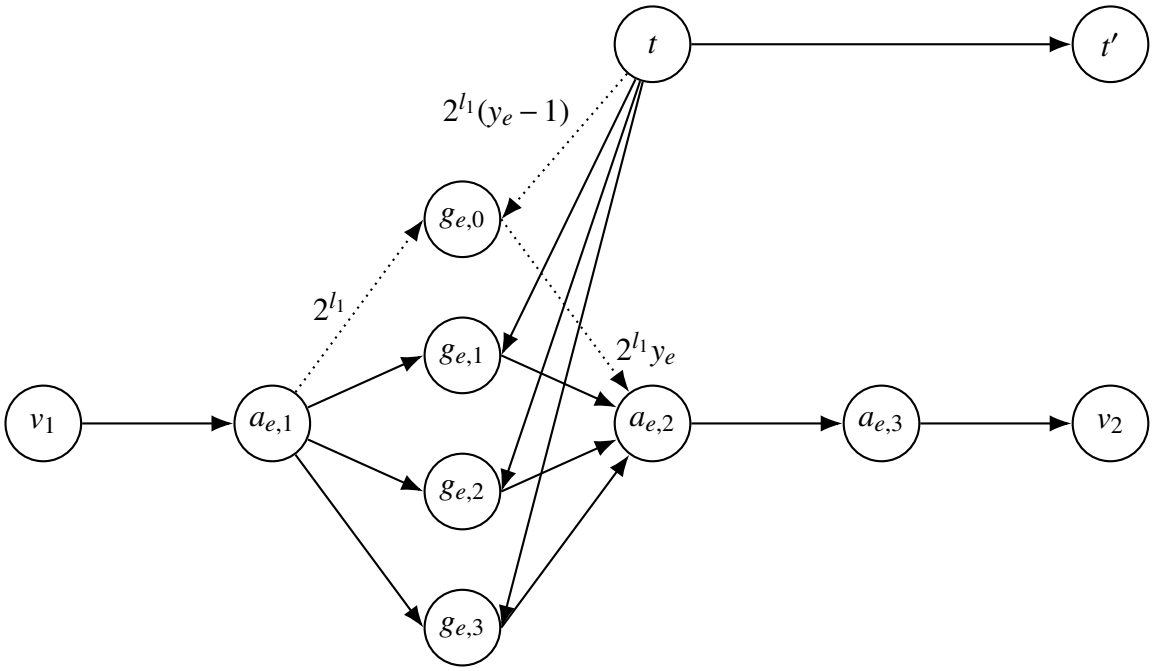

Fig. 5.3.5 Example output graph MFMQ / MCFMQ for the case $z_e = 1$; The edge labels represent the minimum quantities (which coincide with the capacities for these edges) and refer to the dotted edges.

Note that we have in fact created an instance of MFMQ or MCFMQ according to our Definitions 5.2.3 and 5.2.4: All flow multipliers have been removed and there are bounded minimum quantities and capacities on all edges. If the transformation has been applied to an instance of MCFMQ, costs are defined for all edges. Furthermore, there is a dedicated source s and a dedicated sink t', while the demand of all other nodes is 0. The minimum quantities and edge capacities are integral and the edge costs are rational by construction. We denote the instance of MFMQ or MCFMQ that we obtain by applying the above transformation by I' and the underlying graph by $G' = (V', E')$.

We begin our analysis by considering the size of the flow network that we obtain and the number of steps we require to create it: Let $n = |V|$ and $m = |E|$. If $z_e \geq 2$, we create at most $3 + (\log b_e + 1) + (\log_{z_e}(y_e b_e))(\log(z_e - 1) + 1) \leq 4 + \log b_e + (\log y_e + \log b_e)(\log z_e + 1)$ additional nodes for each edge $e \in E$. Furthermore, we create one additional sink node t'. Trivially, this upper bound also holds for the case $z_e = 1$. Hence, $|V'| \leq p_1(n, m, \log y_e, \log b_e, \log z_e)$ for a polynomial function p_1. As G' does not contain any loops or parallel edges, we also have $|E'| \leq p_2(n, m, \log y_e, \log b_e, \log z_e)$ for a polynomial function p_2.

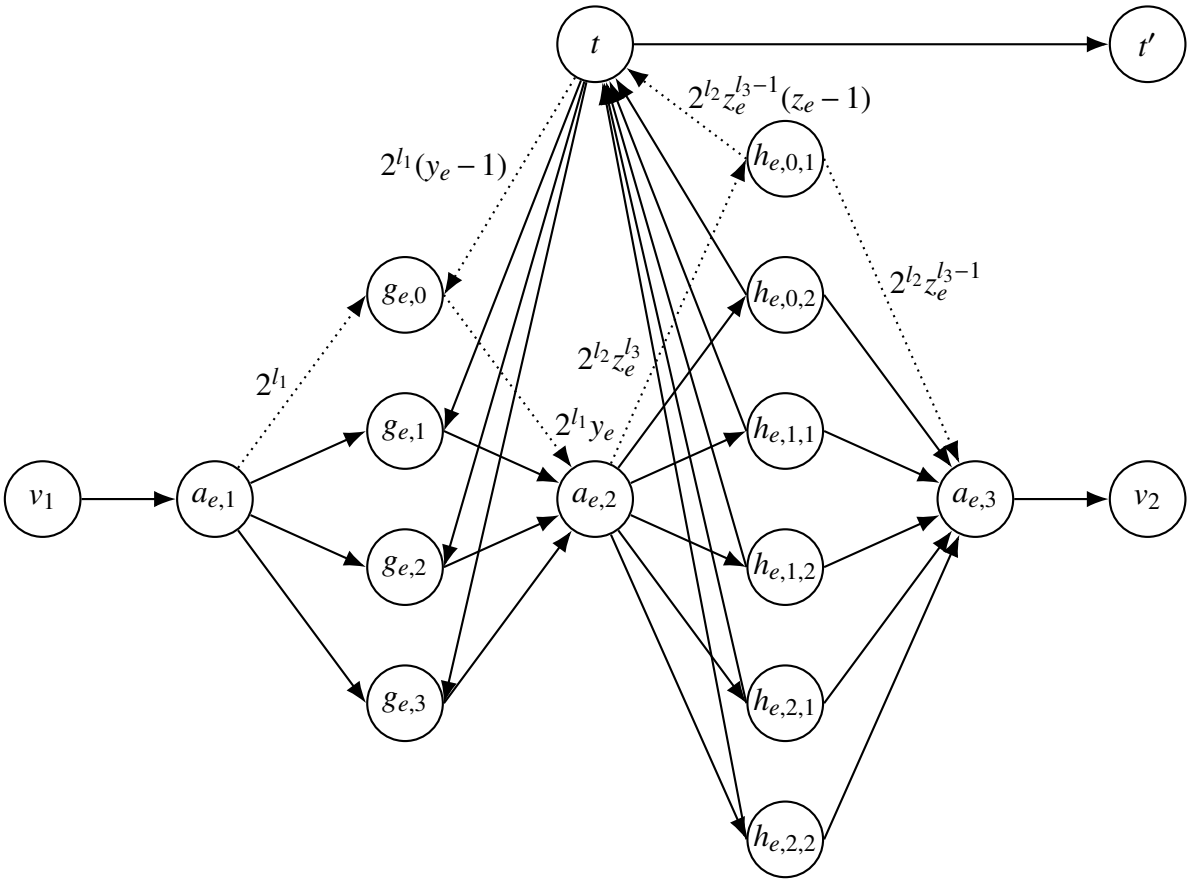

Fig. 5.3.6 Example output graph MFMQ / MCFMQ for the case $z_e \geq 2$; The edge labels represent the minimum quantities (which coincide with the capacities for these edges) and refer to the dotted edges.

We now consider the size of the minimum quantities, capacities and edge costs in G': In order to simplify our analysis, we define the values $q_{\max} := \max_{e \in E} q_e$, $b_{\max} := \max_{e \in E} b_e$, $c_{\max} := \max_{e \in E} c_e$, $y_{\max} := \max_{e \in E} y_e$, $z_{\max} := \max_{e \in E} z_e$, $q'_{\max} := \max_{e' \in E'} q_{e'}$, $b'_{\max} := \max_{e' \in E'} b_{e'}$ and $c'_{\max} := \max_{e' \in E'} c_{e'}$. In addition, we set $\mathrm{val}_{\max} := \max\{q_{\max}, b_{\max}, c_{\max}, y_{\max}, z_{\max}, F\}$ and $\mathrm{val}'_{\max} := \max\{q'_{\max}, b'_{\max}, c'_{\max}, F\}$.

Note that the edge costs in I' are either 0 or correspond to the ones in I. Hence, $c'_{\max} \leq c_{\max}$. Since all edges $e' \in E'$ with $q_{e'} > b_{e'}$ can be removed w.l.o.g., we obtain $q'_{\max} \leq b'_{\max}$. We now consider $b'_{\max}$: By construction, b'_e can attain one of the following values for some $e' \in E'$:

- b_e for some $e \in E$, which is bounded from above by $b_{\max}$ (cf. step 9 of the transformation).

- 2^{l_1}, $2^{l_1}(y_e - 1)$ and $2^{l_1} y_e$ for some $e \in E$, where $l_1 \leq \log b_e$, so that $b'_{\max}$ is bounded from above by $b_{\max} y_{\max}$ (cf. steps 10 - 12 of the transformation).

- $y_e b_e$ for some $e \in E$, which is bounded from above by $b_{\max} y_{\max}$ (cf. steps 13 and 17 of the transformation).

- $2^{l_2} z_e^{l_3}$, $2^{l_2} z_e^{l_3 - 1}(z_e - 1)$ and $2^{l_2} z_e^{l_3 - 1}$ for some $e \in E$, where $l_2 \leq \log(z_e - 1) < \log z_e$ and $l_3 \leq \log_{z_e}(y_e b_e)$, so that $b'_{\max}$ is bounded from above by $b_{\max} y_{\max} z_{\max}$ (cf. steps 14 - 16 of the transformation).

- $\sum_{e \in \delta^+_G(s) \subseteq E} b_e$, which is bounded from above by $m b_{\max}$ (cf. the construction of the edge $e' = (t, t') \in E'$.).

We summarize our observations. Let $|I|$ and $|I'|$ denote the encoding sizes of I and I'.

- From the above analysis we get that $q'_{\max} \leq b'_{\max} \leq m b_{\max} y_{\max} z_{\max}$ and we have already seen that $c'_{\max} \leq c_{\max}$. Hence, $\mathrm{val}'_{\max} \leq m \mathrm{val}^3_{\max}$, i.e. the largest numerical value in I' is bounded by a polynomial in the largest numerical value of I and in $|I|$.
  In particular, the encoding size of $\mathrm{val}'_{\max}$ is polynomially bounded in $|I|$.

- $|V'| \leq p_1(n, m, \log y_e, \log b_e, \log z_e)$ for a polynomial p_1 and
  $|E'| \leq p_2(n, m, \log y_e, \log b_e, \log z_e)$ for a polynomial p_2.
  In particular, the number of edges and nodes (and of the corresponding numerical values such as minimum quantities, edge capacities and edge costs) of G' is bounded by a polynomial in $|I|$.

Combining these two observations, we conclude that I' can be computed in a number of steps that is polynomially bounded in $|I|$.
We now claim that we obtain an optimal solution f to I by solving I' and setting $f(v_1, v_2) \coloneqq f'(v_1, a_{e,1})$ for all $e = (v_1, v_2) \in E$, where f' denotes an optimal solution to I'. We first show that we can transform a feasible solution to I' into a feasible solution to I that has the same objective value.
The following observation will be useful for proving this claim:
Given a feasible flow f' in G' the following holds: Let $e = (v_1, v_2) \in E$. If the flow on one of the edges $(a_{e,1}, g_{e,l_1})$, (t, g_{e,l_1}), $(g_{e,l_1}, a_{e,2})$ is strictly positive for some $e \in E$ and $0 \leq l_1 \leq k_{e,1}$, then the flow on *all* these edges is strictly positive due to the flow conservation constraints. In particular, by construction we have that $y_e \sum_{e' \in \delta^+_{G'}(a_{e,1})} f'_{e'} = \sum_{e' \in \delta^-_{G'}(a_{e,2})} f'_{e'}$. By the same argument we get that $\frac{1}{z_e} \sum_{e' \in \delta^+_{G'}(a_{e,2})} f'_{e'} = \sum_{e' \in \delta^-_{G'}(a_{e,3})} f'_{e'}$. Note that this trivially holds for the case

$z_e = 1$. Combining these observations with the fact that $f'(v_1, a_{e,1}) = \sum_{e' \in \delta^-_{G'}(a_{e,1})} f'_{e'}$ yields the following equality:

$$\begin{aligned}
\mu_e f'(v_1, a_{e,1}) &= \frac{y_e}{z_e} \sum_{e' \in \delta^-_{G'}(a_{e,1})} f'_{e'} = \frac{y_e}{z_e} \sum_{e' \in \delta^+_{G'}(a_{e,1})} f'_{e'} \\
&= \frac{1}{z_e} \sum_{e' \in \delta^-_{G'}(a_{e,2})} f'_{e'} = \frac{1}{z_e} \sum_{e' \in \delta^+_{G'}(a_{e,2})} f'_{e'} \\
&= \sum_{e' \in \delta^-_{G'}(a_{e,3})} f'_{e'} = \sum_{e' \in \delta^+_{G'}(a_{e,3})} f'_{e'} \\
&= f'(a_{e,3}, v_2)
\end{aligned}$$

We show feasibility of f in G and consider the flow integrality constraints first:
Let $v \in V \setminus \{s,t\}$. Then the following holds:

$$\begin{aligned}
f(\delta_G(v)) = f(\delta^-_G(v)) - f(\delta^+_G(v)) &= \sum_{e=(v_1,v) \in \delta^-_G(v)} \mu_e f_e - \sum_{e=(v,v_2) \in \delta^+_G(v)} f_e \\
&\overset{(*)}{=} \sum_{e=(v_1,v) \in \delta^-_G(v)} \mu_e f'(v_1, a_{e,1}) - \sum_{e \in \delta^+_G(v)} f'(v, a_{e,1}) \\
&\overset{(**)}{=} \sum_{e \in \delta^-_G(v)} f'(a_{e,3}, v) - \sum_{e \in \delta^+_G(v)} f'(v, a_{e,1}) \\
&\overset{(***)}{=} \sum_{e' \in \delta^-_{G'}(v)} f'_{e'} - \sum_{e' \in \delta^+_{G'}(v)} f'_{e'} \\
&\overset{(****)}{=} 0
\end{aligned}$$

We obtain the above equalities as follows:

* follows by construction of f.

** follows from the previous observation that $\mu_e f'(v_1, a_{e,1}) = f'(a_{e,3}, v_2)$ for all $e \in E$.

*** follows by construction: For some $v \in V \setminus \{s,t\}$, the set of inbound edges in G' is exactly the set $\left\{(a_{e,3}, v) \in E' \,\middle|\, e \in \delta^-_G(v)\right\}$. Analogously, $\left\{(v, a_{e,1}) \in E' \,\middle|\, e \in \delta^+_G(v)\right\}$ is the set of outbound edges of v.

**** follows from the fact that f' is feasible in G'.

Hence, we have shown that f is feasible regarding the flow integrality constraints for all nodes $v \in V \setminus \{s,t\}$. We now consider the minimum quantities and edge capacities: According

to step 9 of the construction, we have $q_{e'} = q_e$ and $b_{e'} = b_e$ on $e' = (v_1, a_{e,1}) \in E'$ for all $e \in E$. It follows by construction that feasibility of f' implies feasibility of f regarding the minimum quantities and edge capacities. Thus, for GMFMQ we have established feasibility of f. For GMCFMQ we still have to show that f has flow value F. However, it is easy to see that this is in fact the case: As f' is feasible with respect to the flow value F, we have

$$f(\delta_G^+(s)) = \sum_{e \in \delta_G^+(s)} f_e = \sum_{e \in \delta_G^+(s)} f'(s, a_{e,1}) = f'(\delta_{G'}^+(s)) = F.$$

Hence, we have shown feasibility of f for both problems, GMFMQ and GMCFMQ. As a byproduct of the proof we have seen that both solutions have the same flow value. Hence, for GMFMQ both solutions have the same objective value. It is easy to see that this also holds for GMCFMQ: Non-zero edge costs are only defined on the edges $e' = (v, a_{e,1}) \in E'$. By construction we have $f'_{e'} = f'(v, a_{e,1}) = f_e$ and $c'_{e'} = c_e$. Hence, we have shown that we can in fact transform a feasible solution to I' into a feasible solution to I that has the same objective value.

Conversely, we now show that every feasible solution to I induces a feasible solution to I' that has the same objective value:

Let a feasible solution f to I be given and let G' be constructed as above. For every edge $e \in E$ for which $f_e > 0$ find values $\alpha_{e,l_1} \in \{0, 1\}$ for all $0 \leq l_1 \leq k_{e,1}$ so that

$$f_e = \sum_{0 \leq l_1 \leq k_{e,1}} \alpha_{e,l_1} 2^{l_1}.$$

Such a representation exists, since $k_{e,1} = \lfloor \log b_e \rfloor \geq \lfloor \log f_e \rfloor$. Hence, the vector consisting of all α_{e,l_1} is simply the representation of f_e in the numeral system with base 2. Note that $f_e \in \mathbb{N}$ is necessary in order for the above representation to exist. This is the case due to Definition 5.2.1.

If $z_e \geq 2$, find values $\beta_{e,l_3} \in \{0, 1, \ldots, z_e - 1\}$ for all $1 \leq l_3 \leq k_{e,3}$ so that

$$\mu_e f_e = \sum_{1 \leq l_3 \leq k_{e,3}} \beta_{e,l_3} z_e^{l_3 - 1}. \tag{1}$$

By construction we have $k_{e,3} = \left\lfloor \log_{z_e}(y_e b_e) \right\rfloor$. This implies $k_{e,3} - 1 = \left\lfloor \log_{z_e}(y_e b_e) \right\rfloor - 1 = \left\lfloor \log_{z_e}(\mu_e b_e) \right\rfloor \geq \left\lfloor \log_{z_e}(\mu_e f_e) \right\rfloor$. Hence, such a decomposition of $\mu_e f_e$ exists. Note that the vector consisting of all β_{e,l_3} is the representation of $\mu_e f_e$ in the numeral system with base z_e. We require $\mu_e f_e \in \mathbb{N}$ in order for the above representation to exist. Again, this is in fact the case due to Definition 5.2.1.

We decompose each β_{e,l_3} by choosing $\lambda_{e,l_2,l_3} \in \{0,1\}$ for all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$ so that the following equality holds:

$$\beta_{e,l_3} = \sum_{0 \leq l_2 \leq k_{e,2}} \lambda_{e,l_2,l_3} 2^{l_2} \tag{2}$$

By construction we have $k_{e,2} = \lfloor \log(z_e - 1) \rfloor \geq \left\lfloor \log(\beta_{e,l_3}) \right\rfloor$. This implies the above decomposition exists for some $\lambda_{e,l_2,l_3} \in \{0,1\}$. Once again, the vector consisting of all λ_{e,l_2,l_3} is the representation of β_{e,l_3} in the numeral system with base 2.
Combining the equations (1) and (2) yields the following equation:

$$\mu_e f_e = \sum_{\substack{0 \leq l_2 \leq k_{e,2} \\ 1 \leq l_3 \leq k_{e,3}}} \lambda_{e,l_2,l_3} 2^{l_2} z_e^{l_3 - 1}$$

Note that this implies

$$y_e f_e = \sum_{\substack{0 \leq l_2 \leq k_{e,2} \\ 1 \leq l_3 \leq k_{e,3}}} \lambda_{e,l_2,l_3} 2^{l_2} z_e^{l_3}.$$

We are now ready to construct a solution f' to I'. For each $e \in E$ we create the corresponding flow f' in G' as follows:

- Set $f'(v_1, a_{e,1}) \coloneqq f_e$.
- For all $0 \leq l_1 \leq k_{e,1}$: If $\alpha_{e,l_1} = 1$, set $f'(a_{e,1}, g_{e,l_1}) \coloneqq 2^{l_1}$, $f'(t, g_{e,l_1}) \coloneqq 2^{l_1}(y_e - 1)$ and $f'(g_{e,l_1}, a_{e,2}) \coloneqq 2^{l_1} y_e$, else set $f'(a_{e,1}, g_{e,l_1}) \coloneqq f'(t, g_{e,l_1}) \coloneqq f'(g_{e,l_1}, a_{e,2}) \coloneqq 0$.
- Let $z_e = 1$: For $e' = (a_{e,2}, a_{e,3})$ set $f'_{e'} \coloneqq \mu_e f_e = y_e f_e$.
- Let $z_e \geq 2$: For all $0 \leq l_2 \leq k_{e,2}$ and all $1 \leq l_3 \leq k_{e,3}$: If $\lambda_{e,l_2,l_3} = 1$, set $f'(a_{e,2}, h_{e,l_2,l_3}) \coloneqq 2^{l_2} z_e^{l_3}$, $f'(h_{e,l_2,l_3}, t) \coloneqq 2^{l_2} z_e^{l_3 - 1}(z_e - 1)$ and $f'(h_{e,l_2,l_3}, a_{e,3}) \coloneqq 2^{l_2} z_e^{l_3 - 1}$, else set $f'(a_{e,2}, h_{e,l_2,l_3}) \coloneqq f'(h_{e,l_2,l_3}, t) \coloneqq f'(h_{e,l_2,l_3}, a_{e,3}) \coloneqq 0$.
- Set $f'(a_{e,3}, v_2) \coloneqq \mu_e f_e$.

In addition, set $f'(t, t') \coloneqq f(\delta_G^+(s))$.
We claim that the f' is in fact feasible in G'. Once again, we consider the flow conservation constraints first. Let $v \in V' \setminus \{s, t'\}$.

- Let $v \in V \setminus \{t\}$, i.e. v corresponds to one of the nodes in G: By construction, the following holds:

$$\begin{aligned} f'(\delta_{G'}(v)) &= \sum_{e' \in \delta^-_{G'}(v)} f'_{e'} - \sum_{e' \in \delta^+_{G'}(v)} f'_{e'} \\ &= \sum_{e \in \delta^-_G(v)} f'(a_{e,3}, v) - \sum_{e \in \delta^+_G(v)} f'(v, a_{e,1}) \\ &= \sum_{e \in \delta^-_G(v)} \mu_e f_e - \sum_{e \in \delta^+_G(v)} f_e \\ &= 0 \end{aligned}$$

- Let $v = a_{e,1}$ for some $e = (v_1, v_2) \in E$: By construction, we have

$$\begin{aligned} f'(\delta_{G'}(a_{e,1})) &= \sum_{e' \in \delta^-_{G'}(a_{e,1})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(a_{e,1})} f'_{e'} \\ &= f'(v_1, a_{e,1}) - \sum_{0 \le l_1 \le k_{e,1}} f'(a_{e,1}, g_{e,l_1}) \\ &= f_e - \sum_{0 \le l_1 \le k_{e,1}} \alpha_{e,l_1} 2^{l_1} \\ &= f_e - f_e \\ &= 0. \end{aligned}$$

- Let $v = g_{e,l_1}$ for some $e = (v_1, v_2) \in E$ and $0 \le l_1 \le k_{e,1}$:

$$\begin{aligned} f'(\delta_{G'}(g_{e,l_1})) &= \sum_{e' \in \delta^-_{G'}(g_{e,l_1})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(g_{e,l_1})} f'_{e'} \\ &= f'(a_{e,1}, g_{e,l_1}) + f'(t, g_{e,l_1}) - f'(g_{e,l_1}, a_{e,2}) \end{aligned}$$

  If $\alpha_{e,l_1} = 1$, we have $f'(a_{e,1}, g_{e,l_1}) = 2^{l_1}$, $f'(t, g_{e,l_1}) = 2^{l_1}(y_e - 1)$ and $f'(g_{e,l_1}, a_{e,2}) = 2^{l_1} y_e$. Thus, we obtain $f'(\delta_{G'}(g_{e,l_1})) = 0$.
  If $\alpha_{e,l_1} = 0$, we have $f'(a_{e,1}, g_{e,l_1}) = f'(t, g_{e,l_1}) = f'(g_{e,l_1}, a_{e,2}) = 0$, which trivially implies $f'(\delta_{G'}(g_{e,l_1})) = 0$.

- Let $v = a_{e,2}$ for some $e = (v_1, v_2) \in E$:
  For $z_e = 1$ the following holds by construction:

$$\begin{aligned} f'(\delta_{G'}(a_{e,2})) &= \sum_{e' \in \delta^-_{G'}(a_{e,2})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(a_{e,2})} f'_{e'} \\ &= \sum_{0 \le l_1 \le k_{e,1}} f'(g_{e,l_1}, a_{e,2}) - f'(a_{e,2}, a_{e,3}) \\ &= y_e \sum_{0 \le l_1 \le k_{e,1}} \alpha_{e,l_1} 2^{l_1} - y_e f_e \\ &= y_e f_e - y_e f_e \\ &= 0 \end{aligned}$$

  For $z_e \ge 2$ we have

$$\begin{aligned} f'(\delta_{G'}(a_{e,2})) &= \sum_{e' \in \delta^-_{G'}(a_{e,2})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(a_{e,2})} f'_{e'} \\ &= \sum_{0 \le l_1 \le k_{e,1}} f'(g_{e,l_1}, a_{e,2}) - \sum_{\substack{0 \le l_2 \le k_{e,2} \\ 1 \le l_3 \le k_{e,3}}} f'(a_{e,2}, h_{e,l_2,l_3}) \\ &= y_e \sum_{0 \le l_1 \le k_{e,1}} \alpha_{e,l_1} 2^{l_1} - \sum_{\substack{0 \le l_2 \le k_{e,2} \\ 1 \le l_3 \le k_{e,3}}} \lambda_{e,l_2,l_3} 2^{l_2} z_e^{l_3} \\ &= y_e f_e - y_e f_e \\ &= 0 \end{aligned}$$

- Let $v = h_{e,l_2,l_3}$ for some $e = (v_1, v_2) \in E$, $0 \le l_2 \le k_{e,2}$ and $1 \le l_3 \le k_{e,3}$ (i.e. $z_e \ge 2$):

$$\begin{aligned} f'(\delta_{G'}(h_{e,l_2,l_3})) &= \sum_{e' \in \delta^-_{G'}(h_{e,l_2,l_3})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(h_{e,l_2,l_3})} f'_{e'} \\ &= f'(a_{e,2}, h_{e,l_2,l_3}) - f'(h_{e,l_2,l_3}, t) - f'(h_{e,l_2,l_3}, a_{e,3}) \end{aligned}$$

  If $\lambda_{e,l_2,l_3} = 1$, we have $f'(a_{e,2}, h_{e,l_2,l_3}) = 2^{l_2} z_e^{l_3}$, $f'(h_{e,l_2,l_3}, t) = 2^{l_2} z_e^{l_3 - 1}(z_e - 1)$ and $f'(h_{e,l_2,l_3}, a_{e,3}) = 2^{l_2} z_e^{l_3 - 1}$. This implies $f'(\delta_{G'}(g_{e,l_1})) = 0$.
  If $\lambda_{e,l_2,l_3} = 0$, we have $f'(a_{e,2}, h_{e,l_2,l_3}) = f'(h_{e,l_2,l_3}, t) = f'(h_{e,l_2,l_3}, a_{e,3}) = 0$. Thus, we obtain $f'(\delta_{G'}(h_{e,l_2,l_3})) = 0$.

- Let $v = a_{e,3}$ for some $e = (v_1, v_2) \in E$: For $z_e = 1$ the following holds by construction:

$$\begin{aligned} f'(\delta_{G'}(a_{e,3})) &= \sum_{e' \in \delta^-_{G'}(a_{e,3})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(a_{e,3})} f'_{e'} \\ &= f'(a_{e,2}, a_{e,3}) - f'(a_{e,3}, v_2) \\ &= \mu_e f_e - \mu_e f_e \\ &= 0 \end{aligned}$$

  For $z_e \geq 2$ we have

$$\begin{aligned} f'(\delta_{G'}(a_{e,3})) &= \sum_{e' \in \delta^-_{G'}(a_{e,3})} f'_{e'} - \sum_{e' \in \delta^+_{G'}(a_{e,3})} f'_{e'} \\ &= \sum_{\substack{0 \leq l_2 \leq k_{e,2} \\ 1 \leq l_3 \leq k_{e,3}}} f'(h_{e,l_2,l_3}, a_{e,3}) - f'(a_{e,3}, v_2) \\ &= \sum_{\substack{0 \leq l_2 \leq k_{e,2} \\ 1 \leq l_3 \leq k_{e,3}}} \lambda_{e,l_2,l_3} 2^{l_2} z_e^{l_3 - 1} - \mu_e f_e \\ &= \mu_e f_e - \mu_e f_e \\ &= 0. \end{aligned}$$

- Let $v = t$: So far, we have seen that $f'(\delta_{G'}(v)) = 0$ for all $v \in V' \setminus \{s, t, t'\}$. Furthermore, by construction we have $f'(\delta^-_{G'}(t')) = f'(t, t') = f(\delta^+_G(s))$ and $f(\delta^+_G(s)) = f'(\delta^+_{G'}(s))$, which implies $f'(\delta^-_{G'}(t')) = f'(\delta^+_{G'}(s))$. Trivially, we have $\sum_{v \in V'} f'(\delta_{G'}(v)) = 0$.
  All in all, we obtain

$$\begin{aligned} 0 = \sum_{v \in V'} f'(\delta_{G'}(v)) &= \sum_{v \in V' \setminus \{s,t,t'\}} f'(\delta_{G'}(v)) + f'(\delta_{G'}(t')) + f'(\delta_{G'}(s))) + f'(\delta_{G'}(t) \\ = \sum_{v \in V'} f'(\delta_{G'}(v)) &= \sum_{v \in V' \setminus \{s,t,t'\}} f'(\delta_{G'}(v)) + f'(\delta^-_{G'}(t')) - f'(\delta^+_{G'}(s))) + f'(\delta_{G'}(t) \\ = f'(\delta_{G'}(t)) & \end{aligned}$$

Hence, we have shown that f' fulfills the flow conservation constraints for all $v \in V' \setminus \{s, t'\}$. As we have already noted, $f(\delta^+_G(s)) = f'(\delta^+_{G'}(s))$. Hence, if I is an instance of MCFMQ with flow value F, then we immediately obtain $f'(\delta_G(s)) = F$, i.e. f' is feasible with respect to the flow value constraint.

We now consider the minimum quantities and edge capacities. Let $e' \in E'$.

- If $e' = (v_1, a_{e,1})$ for some $e \in E$, we have $q_{e'} = q_e$ and $b_{e'} = b_e$. Since f_e is feasible with respect to q_e and b_e, so is $f'_{e'} = f_e$ with respect to q_e and b_e.
- If $e' \in \{(a_{e,1}, g_{e,l_1}), (t, g_{e,l_1}), (g_{e,l_1}, a_{e,2})\}$ for some $e \in E$ and $0 \leq l_1 \leq k_{e,1}$, we either have $f'_{e'} = q_{e'} = b_{e'}$ or $f'_{e'} = 0$, i.e. the minimum quantity and edge capacity constraints are fulfilled.
- If $z_e = 1$ and $e' = (a_{e,2}, a_{e,3})$ for some $e \in E$, we have $q_{e'} = 0$ and $b_{e'} = y_e b_e$. Hence, $q_{e'} = 0 \leq f'_{e'} = y_e f_e \leq y_e b_e = b_{e'}$.
- If $z_e \geq 2$ and $e' \in \left\{(a_{e,2}, h_{e,l_2,l_3}), (h_{e,l_2,l_3}, t), (h_{e,l_2,l_3}, a_{e,3})\right\}$ for some $e \in E$, $0 \leq l_2 \leq k_{e,2}$ and $1 \leq l_3 \leq k_{e,3}$, we either have $f'_{e'} = q_{e'} = b_{e'}$ or $f'_{e'} = 0$, i.e. the minimum quantity and edge capacity constraints are fulfilled.
- If $e' = (a_{e,3}, v_2)$ for some $e \in E$, we have $q_{e'} = 0$ and $b_{e'} = y_e b_e$. By construction, $f'_{e'} = \mu_e f_e$. Thus, $q_{e'} = 0 \leq f'_{e'} = \mu_e f_e \leq y_e f_e \leq y_e b_e = b_{e'}$, i.e. the minimum quantity and edge capacity constraints are fulfilled.
- If $e' = (t, t')$, we have $q_{e'} = 0$ and $b_{e'} = \sum_{e \in \delta^+_G(s)} b_e$. We have $q_{e'} = 0 \leq f'_{e'} = f(\delta^+_G(s)) = \sum_{e \in \delta^+_G(s)} f_e \leq \sum_{e \in \delta^+_G(s)} b_e = b_{e'}$.

Hence, we have shown that all minimum quantity and edge capacity constraints are fulfilled by f' in G'.
For both objective functions, maximum flow and minimum cost flow, the objective value of f and f' is the same by the same argument as before. Hence, we have shown that every feasible flow f in G induces a feasible flow f' in G' that has the same objective value.
All in all, we have shown that for every feasible solution to I there is a feasible solution to I' that has the same objective value and vice versa. In particular, an optimal solution to I' induces an optimal solution to I. Hence, we obtain an optimal solution to I by determining an optimal solution f' to I' and applying the transformation $f(v_1, v_2) \coloneqq f'(v_1, a_{e,1})$ for all $e = (v_1, v_2) \in E$. □

Remark 5.3.8. Note that the transformation given in Theorem 5.3.7 is a *pseudo-polynomial* transformation as defined in [16]. Hence, solving GMFMQ is at most as hard as solving MFMQ and solving MFMQ is at least as hard as solving GMFMQ. In particular, the transformation retains *strong* NP-hardness of the original problem. On the other hand, since MFMQ is a special case of GMFMQ, solving GMFMQ is at least as hard as solving MFMQ

and solving MFMQ is at most as hard as solving GMFMQ. Hence, we conclude that solving MFMQ is as hard as solving GMFMQ and analogously, solving MCFMQ is as hard as solving GMCFMQ.

We now show how the treewidth of the input graph G and the output graph G' of the transformation given in the proof of Theorem 5.3.7 are related:

Theorem 5.3.9. *Let a graph $G = (V,E)$ with flow multipliers as defined in Definition 5.2.1 be given. Let $G' = (V',E')$ denote the graph that we obtain from applying the transformation given in the proof of Theorem 5.3.7 and let $k = tw(G)$. Then we have $tw(G') \leq \max\{k+1,5\}$.*

Proof. Let a tree-decomposition $D = (\{X_i | i \in I\}, T = (I,F))$ with $\text{width}(D) = k$ of G be given. We show how to create a tree-decomposition $D' = \left(\left\{X'_i \middle| i \in I'\right\}, T' = (I',F')\right)$ of G' with $\text{width}(D') \leq \max\{k+1,5\}$. In order to do so, we apply the following steps:

- For every edge $e = (v_{e,1}, v_{e,2}) \in E$: Find a tree-decomposition $D'_e = \left(\left\{X_i \middle| i \in I_e\right\}, T_e = (I_e, F_e)\right)$ with minimal tree-width of the subgraph of G' induced by the nodes $v_{e,1}, v_{e,2}, a_{e,1}, a_{e,2}, a_{e,3}, g_{e,l_1}, h_{e,l_2,l_3}$ (for all $0 \leq l_1 \leq k_{e,1}$, $0 \leq l_2 \leq k_{e,2}$ and $1 \leq l_3 \leq k_{e,3}$), as defined in Theorem 5.3.7. We denote this induced subgraph by $G'_e = (V'_e, E'_e)$. Note that G'_e is series parallel. Hence, the treewidth is at most 2 (Lemma 1.4.11). By definition, this implies $\max\limits_{i \in I_e} |X_i| \leq 3$ for all $e \in E$.
- For every edge $e = (v_{e,1}, v_{e,2}) \in E$ and $i \in I_e$: Create the set X'_i as follows: $X'_i := X_i \cup \{v_{e,1}, v_{e,2}, t\}$. In particular, we have $\left|X'_i\right| \leq 6$ for all $e \in E$ and $i \in I_e$.
- For every $i \in I$: Create X'_i by setting $X'_i := X_i \cup \{t\}$. In particular, we have $\left|X'_i\right| \leq k+2$.
- For every $e = (v_{e,1}, v_{e,2}) \in E$: Select an arbitrary $i_1 \in I$ for which $v_{e,1}, v_{e,2} \in X'_{i_1}$ and an arbitrary $i_2 \in I_e$. Create a connecting edge $e'_e = (i_1, i_2)$. Note that there is at least one $i_1 \in I$ for which $v_{e,1}, v_{e,2} \in X_{i_1}$ (and thus in X'_{i_1}) since D is a tree-decomposition of G. Denote the set of all these edges by F_c.
- Create a node i_t and the corresponding set $X'_{i_t} := \{t, t'\}$. Hence, $\left|X'_{i_t}\right| = 2$.
- Create an edge $e'_t = (i, i_t)$ for an arbitrary $i \in I$.
- Set $I' := I \bigcup\limits_{e \in E} I_e \bigcup \{i_t\}$
- Set $F' := F \bigcup\limits_{e \in E} F_e \bigcup F_c \bigcup \{e'_t\}$
- Set $D' := \left(\left\{X'_i | i \in I\right\} \bigcup\limits_{e \in E} \left\{X'_i \middle| i \in I_e\right\} \bigcup \left\{X'_{i_t}\right\}, T' = (I',F')\right) = \left(\left\{X'_i | i \in I'\right\}, T' = (I',F')\right)$.

Before we prove the correctness of the claim, we illustrate the construction by the following example: Assume that there are $e_1 = (v_{e_1,1}, v_{e_1,2})$ and $e_2 = (v_{e_2,1}, v_{e_2,2})$ in E. Then, by construction, G' contains the subgraphs G'_{e_1} and G'_{e_2}. The corresponding tree-decompositions are denoted by T_{e_1} and T_{e_2}. Let $i_1 \in I_{e_1}$ and $i_2 \in I_{e_2}$. Then we have $v_{e_1,1}, v_{e_1,2} \in X'_{i_1}$ and $v_{e_2,1}, v_{e_2,2} \in X'_{i_2}$ by construction. The nodes $i'_1, i'_2 \in I$ are chosen so that $v_{e_1,1}, v_{e_1,2} \in X'_{i'_1}$ and $v_{e_2,1}, v_{e_2,2} \in X'_{i'_2}$ and the edges $e'_{e_1} = (i_1, i'_1)$ and $e'_{e_2} = (i_2, i'_2)$ are added to the tree-decomposition of G'. In addition, the node i_t, for which $X'_{i_t} = \{t, t'\}$, and an edge connecting i_t to an arbitrary node from I are added. Figure 5.3.7 illustrates these construction steps:

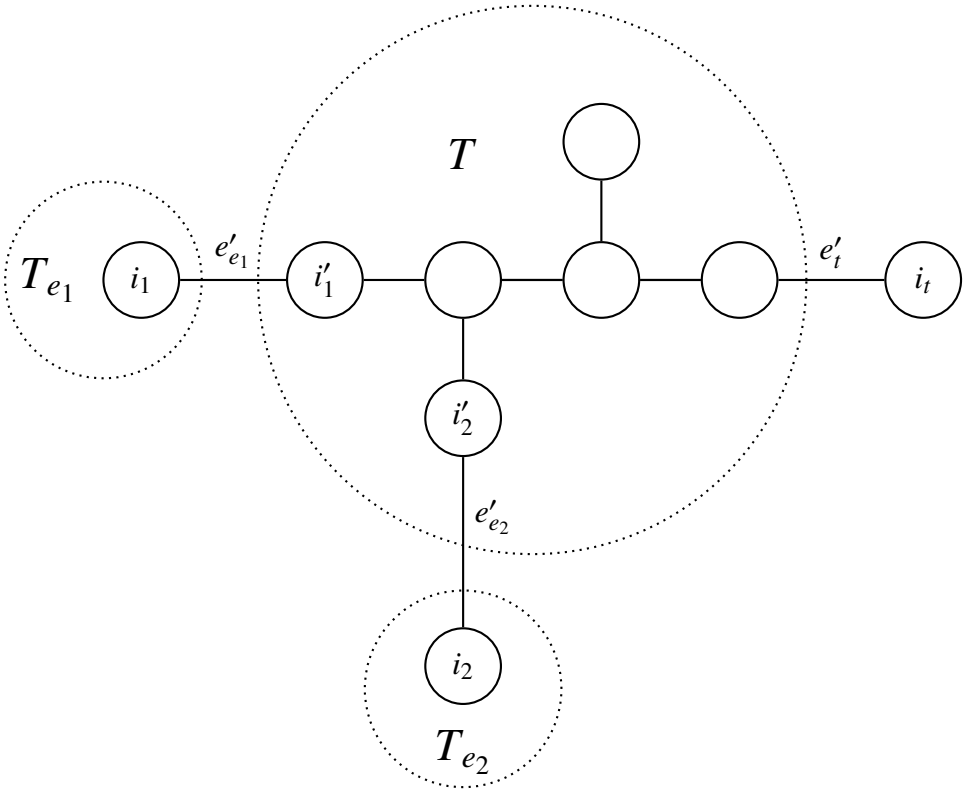

Fig. 5.3.7 Example tree-decomposition of G'; For T_{e_1} and T_{e_2} only the node connecting the respective tree to T is depicted. The dotted circles mark the subtrees T_{e_1}, T_{e_2} and T of T'.

We claim that D' is in fact a feasible tree-decomposition of G':

- Claim 1: T' is a tree.
  The construction of T' begins with $T = (I, F)$, which is a tree by construction. We iteratively add additional trees T_e and every T_e is connected to T by exactly one edge $e'_e \in F_c$. Hence, the resulting graph is also a tree. Finally, the node i_t is added and connected to the graph by the edge e'_t. Thus, T' is in fact a tree.

- Claim 2: $\bigcup_{i \in I'} X'_i = V'$.
  By construction, for an arbitrary node $v \in V$ we have $v \in X'_i$ for some $i \in I \subset I'$. The remaining nodes in $V' \setminus V$ are exactly the nodes $a_{e,1}, a_{e,2}, a_{e,3}, g_{e,l_1}, h_{e,l_2,l_3}$ for all $e \in E$ and for all corresponding values l_1, l_2, l_3, as well as the node t'. By construction, each of the nodes $a_{e,1}, a_{e,2}, a_{e,3}, g_{e,l_1}, h_{e,l_2,l_3}$ is contained in some X'_i for $i \in I_e \subset I'$. In addition, we have $t' \in X'_{i_t}$ and $i_t \in I'$. This shows the claim.

- Claim 3: For all edges $e = (v, w) \in E'$, there exists an $i \in I'$ with $v, w \in X'_i$.
  By construction, we have two types of edges in E': On the one hand, there are edges that are incident to t and on the other hand there are edges that connect two nodes within the same induced subgraph G'_e for some $e \in E$. Note that there is no edge $e = (v, w) \in E'$ where $v \neq t$ and $w \neq t$ and $v \in V(G'_e)$ for some $e \in E$ and $w \notin V(G'_e)$.
  We first consider the case that e is incident to t. W.l.o.g. $v = t$. Then $w \in X'_i$ for some $i \in I'$, as we have already seen. By construction, t is contained in *every* set X'_i so that the claim is fulfilled in this case.
  In the second case we have $v, w \in V(G'_e)$ for some $e \in E$. By construction, D'_e is a tree-decomposition of G'_e. Hence, $v, w \in X'_i$ for some $i \in I_e \subset I'$. Hence, the claim holds.

- Claim 4: For every node $v \in V$ the subset of nodes $\left\{i \in I' | v \in X'_i\right\}$ of T' induces a subtree of T'.
  Let $T'(v)$ denote the induced subgraph. Since T' is a tree, it is clear that $T'(v)$ cannot contain any cycles, so we only have to show connectedness of $T'(v)$. We consider the different cases that can occur:

  - Case 1: $v \in V \setminus \{t\}$
    For some $v \in V \setminus \{t\}$, $e \in E$ and $i \in I_e$ we have that $v \in X'_i$ if and only if $e \in \delta_G(v)$ by construction. In particular, $\left\{i \in I_e | v \in X'_i\right\} = I_e$ if $e \in \delta_G(v)$ and $\left\{i \in I_e | v \in X'_i\right\} = \emptyset$ if $e \notin \delta_G(v)$. For some $v \in V \setminus \{t\}$ and $i \in I$ we have $v \in X_i$ if and only if $v \in X'_i$. Furthermore, $v \notin X'_{i_t} = \{t, t'\}$.
    All in all, we get $\left\{i \in I' | v \in X'_i\right\} = \left\{i \in I \bigcup_{e \in E} I_e \bigcup \{i_t\} \middle| v \in X'_i\right\} = \{i \in I | v \in X_i\} \bigcup_{e \in \delta_G(v)} I_e$.
    Hence, $T'(v)$ is the subgraph of T' induced by the nodes $\{i \in I | v \in X_i\}$ and by the nodes of the trees T_e for which $e \in \delta_G(v)$. Note that the subgraph of T' induced by the nodes of T_e is again T_e.
    Since T is a subgraph of T' and I is the set of nodes of T, the subgraphs of T' and of T induced by $\{i \in I | v \in X_i\}$ are the same. As D is a tree-decomposition of G, the subgraph $T(v)$ of T induced by $\{i \in I | v \in X_i\}$ is in fact a tree.

Connectedness of $T'(v)$ follows if we can show that every T_e for $e \in \delta_G(v)$ is connected to $T(v)$ in $T'(v)$. Let an arbitrary edge $e = (v, v_{e,2}) \in \delta_G(v)$ be given (or $e = (v_{e,1}, v)$, analogously). By construction, there are two nodes $i_1 \in I$ and $i_2 \in I_e$ that are connected by the edge $e'_e = (i_1, i_2)$ in T'. Furthermore, the node i_1 is chosen in such a way that $v \in X'_{i_1}$, which implies that $i_1 \in \left\{i \in I' | v \in X'_i\right\}$. As we have seen, $i_2 \in I_e$ implies $i_2 \in \left\{i \in I' | v \in X'_i\right\}$. Hence, the nodes i_1 and i_2 exist in $T'(v)$ and so does the edge $e'_e = (i_1, i_2)$. Hence, every T_e for $e \in \delta_G(v)$ is in fact connected to $T(v)$ and thus $T'(v)$ is connected.

- Case 2: $v = t$
  By construction we have $v \in X'_i$ for all $i \in I'$. Hence, $T'(v) = T'$, which is a tree by construction.
- Case 3: $v = t'$
  By construction t' is only included in X'_{i_t}. Thus, $T'(v)$ contains only the node i_t.
- Case 4: $v \in V'_e \setminus V$ for some $e \in E$
  Note that $V'_e \setminus V$ consists of the nodes $a_{e,1}$, $a_{e,2}$, $a_{e,3}$, g_{e,l_1}, h_{e,l_2,l_3} for $0 \leq l_1 \leq k_{e,1}$, $0 \leq l_2 \leq k_{e,2}$ and $1 \leq l_3 \leq k_{e,3}$, since $v_{e,1}, v_{e,2} \in V$.
  These nodes only occur in the respective subgraph G'_e and $v \in X'_i$ implies $i \in I_e$. Hence, $\left\{i \in I' | v \in X'_i\right\} = \left\{i \in I_e | v \in X'_i\right\}$. Furthermore, for $v \in V'_e \setminus V$ and some $i \in I_e$ we have $v \in X_i$ if and only if $v \in X'_i$. Combining these two observations we get $\left\{i \in I' | v \in X'_i\right\} = \{i \in I_e | v \in X_i\}$. Hence, the subgraph $T_e(v)$ of T_e induced by the nodes $\{i \in I_e | v \in X_i\}$ is exactly $T'(v)$. Since D_e is a tree-decomposition of G'_e, the subgraph $T_e(v)$ (and thus $T'(v)$) is in fact a tree.

We have already seen that $\left|X'_i\right| \leq k+2$ if $i \in I_e$ for some $e \in E$, $\left|X'_i\right| \leq 6$ for $i \in I$ and $\left|X'_{i_t}\right| = 2$. Hence, $\mathrm{tw}(G') \leq \mathrm{width}(D') = \max\limits_{i \in I'} \left|X'_i\right| - 1 \leq \max\{k+1, 5\}$. □

In our definition of GMFMQ and GMCFMQ, respectively, we have restricted our analysis to instances for which a finite upper capacity bound is defined on all edges. We conclude this section by showing how instances of MFMQ and MCFMQ, where the upper capacity bound for some edges is undefined (i.e. $b_e = \infty$) can be handled:
Let an instance I of MFMQ or MCFMQ be given and set $m \coloneqq |E|$. For MFMQ we can check in polynomial time if there is an s-t-path without any edge capacity constraints. For MCFMQ we can check in polynomial time whether there is a cycle without any edge capacity constraints and for which the sum of edge costs is negative. If so, the objective value of the respective instance of MFMQ or MCFMQ is unbounded. Else, let E_b be the set of edges with a finite upper capacity constraints and let E_u be the edges for which $b_e = \infty$. For MFMQ

set $B := \max\left\{\max_{e\in E_b} b_e, \max_{e\in E_u} q_e\right\}$. For MCFMQ set $B := \max\left\{\max_{e\in E_b} b_e, \max_{e\in E_u} q_e, F\right\}$. Note that in both cases B is well-defined and finite: If we had $b_e = \infty$ for all $e \in E$, then MFMQ would be unbounded. By definition of MCFMQ, a finite flow value F is always defined. Set $b_e := mB$ for all $e \in E_u$. We call this instance I' and claim that the optimal objective values of I and I' are identical. Since we are adding additional constraints in order to create I', an optimal solution to I' cannot be better than an optimal solution to I. Hence, it is sufficient to show that an optimal solution to I' is at least as good as an optimal solution to I.

Let an optimal solution f to I be given. We use the flow f as a baseline to create an updated flow f' which is a feasible solution to I' and which has the same objective value as f. We initialize f' by setting $f' := f$ and update the flow in order to obtain a flow that is feasible regarding the updated edge capacities. The given flow f can be decomposed into at most m cycles and s-t-paths. Let such a decomposition be given and let E_P denote the set of edges on a path (or cycle) P of the decomposition and f_P the flow value on that path (or cycle) of the decomposition. Note that if the decomposition of f contains a cycle C for which $f_C > B$, then $E_C \subseteq E_u$. For all cycles C with $f_C > B$ we created an updated flow by setting $f'_C := B$. For MFMQ this does not affect the objective value, since the s-t-flow remains unchanged. For MCFMQ there is no cycle C for which the sum of the edge costs is negative and for which $E_C \subseteq E_u$. Hence, reducing the flow on C does not increase the objective value.

By construction $f'_e \geq q_e$ for all $e \in E$ and $f'_e \leq b_e$ for all $e \in E_b$.

Note that we have reduced the flow to at most B on each of the cycles of a given decomposition. However, an edge might occur in multiple paths and cycles of a composition, so that we still have to show that $f'_e \leq b_e$ for all $e \in E_u$. For MFMQ we have that every s-t-path contains at least one edge $e \in E_b$ by assumption. Hence $f'_P \leq B$ for all s-t-paths P. For MCFMQ we have that $f'_P \leq F \leq B$ for all s-t-paths P. Now consider an arbitrary edge $e \in E_u$. The edge is part of at most m cycles or s-t-paths of the above decomposition of f, each of which has a flow value at most B in the updated solution f'. Hence, the flow on the respective edge is at most mB, which implies $f'_e \leq mB = b_e$ for all $e \in E_u$.

Note that the above procedure works for MRFMQ and MCRFMQ as well.

5.4 Conclusion

In this chapter we have provided definitions for several flow problems with minimum quantities, some of which had already been considered in earlier publications in similar form. We have pointed out the differences between the definitions used in these publications and we have given an overview of the respective results. Based on this we were able to provide

further insights regarding flow problems with minimum quantity constraints. In order to do so, it proved to be helpful that we were able to show that MFMQ and MCFMQ can be reduced to MWECBMMQ (Theorem 5.3.1).
For the following problems we could show that, unless P=NP, there cannot be any polynomial-time bicriteria approximation algorithm, even on series parallel or on bipartite graphs:

- MCFMQ and MCRFMQ (Theorem 5.3.4)
  The proof of the theorem also shows NP-hardness of the problems, however, this has already been shown before.

Conversely, in the following case, we could show pseudo-polynomial solvability:

- MFMQ and MCFMQ on graphs for which the treewidth is at most r for fixed $r \geq 1$ (Corollary 5.3.2)

The following problem can be solved in polynomial time:

- MFMQ on series parallel graphs if the number of different minimum quantities is bounded by r for fixed $r \geq 1$ (Corollary 5.3.3)

In addition, we could establish the following approximability result:

- For $\epsilon > 0$ there is a polynomial $(1, 1+\epsilon)$-approximation algorithm for MRFMQ on series parallel graphs (Theorem 5.3.6)
  In order to show the claim, we first had to show that for every instance of MRFMQ there is an integral optimal solution (Lemma 5.3.5).

In addition to the above results, we could show a polynomial-time transformation between generalized flow problems with minimum quantities and (non-generalized) flow problems with minimum quantities, which implies that both classes of problems have the same computational complexity (Theorem 5.3.7). As we have pointed out in Remark 5.3.8, the transformation given in Theorem 5.3.7 is in fact a *pseudo-polynomial* transformation. Hence it retains *strong* NP-hardness.
Finally, in Theorem 5.3.9 we could show that if the transformation given in Theorem 5.3.7 is applied to a graph for which the treewidth is bounded by a constant k, then the treewidth of the output graph is bounded by $\max\{k+1, 5\}$.

Chapter 6

Conclusion

6.1 Summary of the results

In this thesis we have considered the application of minimum quantity constraints to four classes of problems: Bin packing problems, scheduling problems, matching problems and flow problems. For each of these classes of problems we have provided a summary of the definitions and results that existed to date. We have defined additional problems with minimum quantity constraints that, to the best of our knowledge, had not been considered before. In particular, we have applied minimum quantity constraints to the maximum-weight b-matching problem and to open shop scheduling problems. A detailed summary of our contribution regarding the respective problem class can be found at the end of each chapter. Hence, we will not repeat the results in detail in this chapter.

As expected based on previous results, we could show that most problems become NP-hard if minimum quantity constraints are added. Hence, we have restricted the problems in order to find special cases that can be solved in polynomial or even linear time. For problems defined on graphs such as matching and flow problems, applying a fixed upper bound on the treewidth of the graph has proven to be helpful. For scheduling problems that allow preemption and that can be solved in polynomial time we have shown that these problems remain polynomially solvable even with identical minimum quantities. For several problems we were able to provide algorithms with a pseudo-polynomial running time.

In addition, we have considered approximability of the problems: For most problems we could show that, unless P=NP, there cannot be any polynomial-time approximation algorithm or that the approximation ratio cannot exceed certain bounds. Hence, we have considered bicriteria approximation algorithms that allow the constraints of the problem to be violated up to a certain degree. This approach has proven to be very helpful and we were able to provide a polynomial-time bicriteria approximation algorithm for at least one problem of

each of the four problem classes that we have considered. For problems defined on graphs, the class of series parallel graphs supports this approach very well. In order to ensure a polynomial running time of the algorithms, we have used different technical approaches such as rounding, clustering and restricting the set of partial solutions to certain representatives. All in all, this thesis is a comprehensive summary of the current state of research on minimum quantity constraints and can be used as a starting point for further research.

6.2 Ideas for future research

In this section we would like to give some ideas that might be covered by future research. First of all, the idea of minimum quantity constraints could obviously be applied to new classes of problems or to additional problems within the problem classes that we have covered in this thesis. In particular, further research might be dedicated to applying minimum quantity constraints to scheduling problem by considering additional combinations of machine environments, objective functions and other constraints. Moreover, other shop scheduling problems, such as flow shop or job shop scheduling problems, could be defined and analyzed. The definition of bin packing problems with minimum quantity constraints could be extended to *classes* of items: For example, the model could be defined in such a way that the items *of each class* that are packed into a specific bin have to fulfill the respective minimum quantity constraint on the bin. The bin capacity could be defined as an upper bound on the sum of the sizes of *all* items that are packed into the respective bin. Different minimum quantities could be defined on the same bin for the different classes of items. Analogously, *classes of jobs* could be added to the existing definitions of scheduling problems. We also assume that there is some potential for further research regarding multi-commodity flow problems.
As we have seen, bicriteria approximation algorithms are a helpful approach for tackling problems with minimum quantities. Hence, trying to develop additional polynomial-time bicriteria approximation algorithms seems like a promising topic for future research. In particular, we have shown that for every $\epsilon > 0$ there is a polynomial-time $(1+\epsilon, 1+\epsilon)$-approximation algorithm for $Pq||C_{\max}$. It remains open, whether a similar result is possible for $Pq||\sum C_i$.
Throughout this thesis we have focused on the theoretical analysis of the problems. Hence, designing efficient implementations of the algorithms presented in this thesis and measuring their performance in practical applications might be an interesting topic for future research. In addition, there might be algorithms with exponential running time that still prove to be efficient in practice.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., USA, 1993. ISBN 013617549X.

[2] Ashwin Arulselvan, Ágnes Cseh, Martin Groß, David F. Manlove, and Jannik Matuschke. Matchings with Lower Quotas: Algorithms and Complexity. *Algorithmica*, 80 (1):185–208, 2016. doi: 10.1007/s00453-016-0252-6.

[3] Péter Biró, Tamás Fleiner, Robert W. Irving, and David Manlove. The College Admissions problem with lower and common quotas. *Theoretical Computer Science*, 411 (34-36):3136–3153, 2010. doi: 10.1016/j.tcs.2010.05.005.

[4] Hans L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11(1-2): 1–21, 1993.

[5] Hans L. Bodlaender. A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 226–234, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897915917. doi: 10.1145/167088.167161.

[6] Hans L. Bodlaender and Babette de Fluiter. Parallel Algorithms for Series Parallel Graphs. In *Algorithms — ESA '96*, pages 277–289, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70667-0.

[7] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008. ISSN 0010-4620. doi: 10.1093/comjnl/bxm037.

[8] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, USA, 1999. ISBN 089871432X.

[9] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2001. ISBN 3540415106.

[10] Katarína Cechlárová and Tamás Fleiner. Pareto optimal matchings with lower quotas. *Mathematical Social Sciences*, 88:3–10, 2017. doi: 10.1016/j.mathsocsci.2017.03.007.

[11] Johanne Cohen and Fanny Pascual. Scheduling Tasks from Selfish Multi-tasks Agents. In *Euro-Par 2015: Parallel Processing - 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings*, pages 183–195, 2015. doi: 10.1007/978-3-662-48096-0_15.

[12] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998. ISBN 047155894X.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844.

[14] Vithya Ganeshan. Fast Method for Maximum-Flow Problem with Minimum-Lot Sizes, 2015.

[15] Michael R. Garey and David S. Johnson. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM*, 25:499–508, July 1978. ISSN 0004-5411. doi: 10.1145/322077.322090.

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710455.

[17] Fred W. Glover and Darwin D. Klingman. On the equivalence of some generalized network problems to pure network problems. *Mathematical Programming*, 4(1):269–278, Dec 1973. ISSN 1436-4646. doi: 10.1007/BF01584670.

[18] Teofilo Gonzalez and Sartaj Sahni. Open Shop Scheduling to Minimize Finish Time. *Journal of the ACM*, 23(4):665–679, October 1976. ISSN 0004-5411. doi: 10.1145/321978.321985.

[19] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[20] Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of discrete mathematics*, 5(2):287–326, 1979. doi: 10.1016/S0167-5060(08)70356-X.

[21] Philipp Hahn. *Scheduling with Minimum Quantities*. Diploma thesis, University of Kaiserslautern, 2013.

[22] Leslie A. Hall and David B. Shmoys. Approximation Schemes for Constrained Scheduling Problems. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 134–139, 1989. doi: 10.1109/SFCS.1989.63468.

[23] Dag Haugland, Mujahed Eleyat, and Magnus L. Hetland. The Maximum Flow Problem with Minimum Lot Sizes. In *Computational Logistics - Second International Conference, ICCL 2011, Hamburg, Germany, September 19-22, 2011. Proceedings*, pages 170–182, 2011. doi: 10.1007/978-3-642-24264-9_13.

[24] Dorit S. Hochbaum and David B. Shmoys. Using Dual Approximation Algorithms for Scheduling Problems Theoretical and Practical Results. *Journal of the ACM*, 34(1): 144–162, January 1987. ISSN 0004-5411. doi: 10.1145/7531.7535.

[25] Michael Hopf. *Network Flows with Minimum Quantities*. Master thesis, University of Kaiserslautern, 2013.

[26] Naoyuki Kamiyama. A note on the serial dictatorship with project closures. *Operations Research Letters*, 41(5):559–561, 2013. doi: 10.1016/j.orl.2013.07.006.

[27] Viggo Kann. Maximum Bounded 3-dimensional Matching is MAX SNP-complete. *Information Processing Letters*, 37(1):27–35, 1991. doi: 10.1016/0020-0190(91)90246-E.

[28] Sven O. Krumke. *On the Approximability of Location and Network Design Problems*. PhD thesis, Julius Maximilian University of Würzburg, 1997.

[29] Sven O. Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag, 2005. ISBN 978-3-519-00526-1. doi: 10.1007/978-3-322-92112-3.

[30] Sven O. Krumke and Alexander Sieber. Matching problems with minimum quantity constraints (Abstract). *Conference program of the conference OR 2018 in Brussels*, 2018.

[31] Sven O. Krumke and Clemens Thielen. Minimum cost flows with minimum quantities. *Information Processing Letters*, 111(11):533–537, 2011. doi: 10.1016/j.ipl.2011.03.007.

[32] Sven O. Krumke and Clemens Thielen. Erratum to "Minimum cost flows with minimum quantities" [Information Processing Letters 111 (11) (2011) 533-537]. *Information Processing Letters*, 112(13):523–524, 2012. doi: 10.1016/j.ipl.2012.03.016.

[33] Sven O. Krumke and Clemens Thielen. The Generalized Assignment Problem with Minimum Quantities. *European Journal of Operational Research*, 228(1):46–55, 2013. doi: 10.1016/j.ejor.2013.01.027.

[34] Sven O. Krumke and Christiane Zeck. Generalized Max Flow in Series-Parallel Graphs. *Discrete Optimization*, 10:155–162, 05 2013. doi: 10.1016/j.disopt.2013.01.001.

[35] Sven O. Krumke, Madhav V. Marathe, Hartmut Noltemeier, Venkatesh Radhakrishnan, Sekharipuram S. Ravi, and Daniel J. Rosenkrantz. Compact Location Problems. *Theoretical Computer Science*, 181(2):379–404, 1997. doi: 10.1016/S0304-3975(96)00304-0.

[36] Sven O. Krumke, Madhav V. Marathe, Hartmut Noltemeier, Ramamoorthi Ravi, and Sekharipuram S. Ravi. Approximation Algorithms for Certain Network Improvement Problems. *Journal of Combinatorial Optimization*, 2(3):257–288, 1998. doi: 10.1023/A:1009798010579.

[37] Jacques Labetoulle, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Preemptive Scheduling of Uniform Machines Subject to Release Dates. In *Progress in Combinatorial Optimization*, pages 245 – 261. Academic Press, 1984. ISBN 978-0-12-566780-7. doi: 10.1016/B978-0-12-566780-7.50020-9.

[38] Robert McNaughton. Scheduling with Deadlines and Loss Functions. *Management Science*, 6(1):1–12, October 1959. ISSN 0025-1909. doi: 10.1287/mnsc.6.1.1.

[39] Daniel Monte and Norovsambuu Tumennasan. Matching with quorums. *Economics Letters*, 120(1):14 – 17, 2013. ISSN 0165-1765. doi: 10.1016/j.econlet.2013.03.007.

[40] Erez Petrank. The Hardness of Approximation: Gap Location. *Computational Complexity*, 4(2):133–157, 1994. ISSN 1420-8954. doi: 10.1007/BF01202286.

[41] William R. Pulleyblank. Edmonds, Matching and the Birth of Polyhedral Combinatorics. *Documenta Mathematica*, 01 2012.

[42] Ramamoorthi Ravi, Madhav V. Marathe, Sekharipuram S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt. Many birds with one stone: multi-objective approximation algorithms. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 438–447, 1993. doi: 10.1145/167088.167209.

[43] Petra Scheffler. A Practical Linear Time Algorithm for Disjoint Paths in Graphs with Bounded Tree-width. Preprint 396, Technische Universität Berlin, Institut für Mathematik, 1994.

[44] Hans Georg Seedig. Network Flow Optimization with Minimum Quantities. In *Operations Research Proceedings 2010, Selected Papers of the Annual International Conference of the German Operations Research Society, Universität der Bundeswehr, München, September 1-3, 2010*, pages 295–300, 2010. doi: 10.1007/978-3-642-20009-0_47.

[45] Alexander Sieber. *Optimization Problems with Minimum Quantity Constraints*. Diploma thesis, University of Kaiserslautern, 2012.

[46] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi: 10.1002/nav.3800030106.

[47] Clemens Thielen and Stephan Westphal. Complexity and approximability of the maximum flow problem with minimum quantities. *Networks*, 62(2):125–131, 2013. doi: 10.1002/net.21502.

[48] Xiaoyan Zhu, Qi Yuan, Alberto Garcia-Diaz, and Liang Dong. Minimal-cost network flow problems with variable lower bounds on arc flows. *Computers & Operations Research*, 38(8):1210–1218, 2011. doi: 10.1016/j.cor.2010.11.006.

Appendix

Curriculum Vitae

Juni 2007	Schulabschluss (Abitur) Christian-Wirth-Schule, Usingen
Oktober 2007 - Dezember 2012	Studium der Wirtschaftsmathematik (Diplom) Technische Universität Kaiserslautern, Fachbereich Mathematik
Dezember 2012	Hochschulabschluss (Diplom Wirtschaftsmathematiker) Technische Universität Kaiserslautern, Fachbereich Mathematik
September 2016 - Februar 2020	Promotion in Mathematik Technische Universität Kaiserslautern, Fachbereich Mathematik

June 2007	High school graduation (Abitur) Christian-Wirth-Schule, Usingen
October 2007 - December 2012	Studies in business mathematics (Diplom Wirtschaftsmathematik) University of Kaiserslautern, Department of Mathematics
December 2012	Graduation in business mathematics (Diplom Wirtschaftsmathematik) University of Kaiserslautern, Department of Mathematics
September 2016 - February 2020	Doctoral candidate in mathematics University of Kaiserslautern, Department of Mathematics