Dissertation

# On the Complexity, Approximation and Modeling Aspects of Special Partition Problems

by

Bernhard Reinbold

Full imprint approved by the Fakultät für Informatik of the Universität der Bundeswehr München to obtain the academic degree of Doctor rerum naturalium (Dr. rer. nat.).

This dissertation was submitted to the Universität der Bundeswehr München and accepted by the Fakultät für Informatik on March 27th, 2014. The oral examination was held on October 22nd, 2014.

**Abstract**

In many practical applications partition problems play an important role. In most cases in this context a target function and a basic set are given. The basic set has to be divided into subsets, while certain constraints have to be satisfied. To define the mostly linear target functions and constraints weight functions are used. These assign a constant value to each element of the basic set. In this work a partition problem is analyzed in which the weights are not constant but also depend on the partition itself. Especially, the complexity in relation with monotonic properties of the weight function is in the focus of the research. Further, another partition problem with piecewise linear convex target function is defined and modeled as integer linear program. Additionally, heuristics are developed that solve the problem in short time.

## Abstract (German)

In vielen praktischen Anwendungen spielen Partitionsprobleme eine wichtige Rolle. In den meisten Fällen sind in diesem Zusammenhang eine Zielfunktion und eine Grundmenge gegeben. Die Grundmenge muss in Teilmengen unterteilt werden, wobei bestimmte Nebenbedingungen eingehalten werden müssen. Zur Definition der meist linearen Zielfunktionen und Nebenbedingungen werden Gewichtsfunktionen verwendet. Diese ordnen jedem Element der Grundmenge einen konstanten Wert zu. In dieser Arbeit wird ein Partitionsproblem analysiert, in dem die Gewichte nicht konstant sind, sondern auch von der Partition selbst abhängen. Vor allem die Komplexität in Abhängigkeit von Monotonieeigenschaften der Gewichtsfunktion steht im Mittelpunkt der Untersuchungen. Außerdem wird ein weiteres Partitionsproblem mit stückweise linearer konvexer Zielfunktion definiert und als ganzzahliges lineares Programm modelliert. Darüber hinaus werden Heuristiken entwickelt, die das Problem in kurzer Zeit lösen.

**Acknowledgements**

First of all, I want to thank Prof. Dr. Andreas Brieden for all his support and time in not only scientific but also personal discussions, for the freedom he gave me in my work and the many cheerful moments. Special thanks also go to Prof. Dr. Peter Gritzmann for the many enlightening conversations and insights.
Further, I want to thank all professors at the faculty of computer science for the warm welcome. Big thanks also go to Prof. Dr. Stefan Pickl who accepted to be my second advisor without hesitation.
Very special thanks go to Kerstin for all the listening and the encouraging words that were so helpful on many occasions. At this point, I also want to thank my family for all the support and help.
Last but not least, special thanks go to my friends and colleagues for many constructive discussions and the sympathy during the last years.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

## Abbreviations

| | |
|---|---|
| **CJAP** | Compact Convex Job Assignment Problem |
| **CMJAP** | Compact Convex Movable Job Assignment Problem |
| **CSC** | Constrained Minimum-$k$-Star Clustering |
| **FPP** | Feasible Partition Problem |
| **ILP** | Integer Linear Program |
| **JAP** | Convex Job Assignment Problem |
| **JSP** | Job-Shop Problem |
| **MCSP** | Minimum Common String Partition Problem |
| **MinSumPP** | MinSum Partition Problem |
| **MinMaxPP** | MinMax Partition Problem |
| **MJAP** | Convex Movable Job Assignment Problem |
| **NPP** | Number Partition Problem |
| **OCCP** | Optimum Cost Chromatic Partition Problem |
| **QAP** | Quadratic Assignment Problem |
| **SSP** | Subset Sum Problem |
| **TSP** | Travelling Salesman Problem |
| **VRP** | Vehicle Routing Problem |

# Nomenclature

| | |
|---|---|
| $\mathbb{N}$ | natural numbers: $\{1, 2, \ldots\}$ |
| $\mathbb{N}_0$ | natural numbers including 0: $\{0, 1, 2, \ldots\}$ |
| $\mathbb{R}_{\geq a}$ | real numbers larger or equal than $a$: $[a, \infty)$ |
| $\mathbb{R}_{>a}$ | real numbers larger than $a$: $(a, \infty)$ |
| $\mathbb{R}^+$ | positive real numbers: $(0, \infty)$ |
| $\mathbb{R}_0^+$ | nonnegative real numbers: $[0, \infty)$ |

## Chapter 3

| | |
|---|---|
| $V$ | basic set |
| $\mathcal{P}(V)$ | power set of $V$ |
| $\mathcal{W}$ | partition of $V$ |
| $W$ | partitioning set of $\mathcal{W}$ |
| $\mathcal{W}^x$ | partition of $V$ induced by incidence vector $x$ |
| $\mathcal{W}_{(v,j)}$ | partition in which $v$ is moved to $j$ |
| $\mathcal{W}_{(v,w)}$ | partition in which $v$ and $w$ are switched |
| $p(W, v)$ | subset function on $V$ |
| $P(W)$ | cumulated subset function on $V$ |
| $s_p(\mathcal{W})$ | sum subset value of $\mathcal{W}$ |
| $m_p(\mathcal{W})$ | maximum subset value of $\mathcal{W}$ |
| $g_s(\mathcal{W}, \mathcal{W}')$ | function comparing to partitions with respect to $s_p$ |
| $g_m(i, \mathcal{W}, \mathcal{W}')$ | function comparing to partitions with respect to $m_p$ |

| | |
|---|---|
| $M(\mathcal{W})$ | set of possible moves for $\mathcal{W}$ |
| $S(\mathcal{W})$ | set of possible switches for $\mathcal{W}$ |
| $I_{\mathcal{W}}$ | maxsets indices of $\mathcal{W}$ |
| $M(\mathcal{W}; I)$ | set of possible moves for $\mathcal{W}$ affecting $I$ |
| $S(\mathcal{W}; I)$ | set of possible switches for $\mathcal{W}$ affecting $I$ |

## Chapter 4

| | |
|---|---|
| $j$ | job |
| $k$ | movable job |
| $s$ | starting time of a job |
| $d$ | duration time of a (movable) job |
| $s^{\min}$ | lower bound for the starting time |
| $s^{\max}$ | upper bound for the starting time |
| $J$ | set of jobs |
| $K$ | set of movable jobs |
| $O_J$ | set of overlapping subsets of $J$ |
| $\rho_J$ | size of a maximal cardinality overlapping subset of $J$ |
| $G_J$ | job graph for $J$ |
| $\delta_G(v)$ | neighborhood of vertex $v$ in graph $G$ |
| $M$ | set of machines |
| $T$ | set of machine types |
| $\hat{t}$ | function assigning machines to a machine types |
| $r$ | restriction matrix |

| | |
|---|---|
| $f$ | production limit |
| $c$ | cost vector |
| $a$ | solution assignment function |
| $x$ | variable assigning (movable) job to machine (type) |
| $z$ | variable counting number of machines of one type |
| $y$ | variable to model convex target function |
| $s$ | variable for starting time of movable job |
| $ss$ | variable deciding if job starts before other job starts |
| $es$ | variable deciding if job ends before other job starts |
| $xss$ | product of $x$ and $ss$ |
| $xes$ | product of $x$ and $es$ |
| $J_t$ | jobs processed by machine type $t$ |
| $J_{s,t}$ | jobs induced by $s$ and processed by machine type $t$ |
| $u$ | variable assigning job to machine |
| $v$ | variable for maximal processing time |

# Introduction

Partition problems are one of the most common and basic problems in discrete optimization. Besides stand-alone variants they often occur as subproblems in larger scale problem settings. In general all partition problems share that an arbitrary finite basic set has to be divided into a number of subsets.
Basically there are two types of partition problems. The first are decision type problems, where a partition has to be found that satisfies some condition or a set of conditions. The second are minimization (or maximization) type problems, where a partition has to be found that minimizes (or maximizes) some target function. The conditions and target functions are often expressed on the basis of different weight functions. These weight functions assign values to elements or tuples of elements of the basic set. Furthermore, in most cases the two problem types are mixed. This means a minimal partition with respect to the target function has to be found among the set of feasible partitions that fulfill the given conditions.

## Example partition problems

In the following a large variety of partition problems and the state of research in this field are presented.

The most elementary decision type partition problem is the Number Partition Problem (NPP). In NPP the basic set is a finite set of natural numbers. The goal is to find a partition of this set into two subsets such that the sums of the elements in both subsets are equal. According to Garey and Johnson [37] already this elementary problem is $\mathbb{NP}$-complete.
In particular, in this work NPP is very important since the newly introduced special partition problems are closely related. Especially, this becomes obvious in the proofs of the corresponding complexity results. In these proofs the $\mathbb{NP}$-hardness of NPP is transferred to the according partition problem.

Partition problems very often arise in the field of graph theory. Holyer [46], for example, studied the problem to partition the edges of a graph such that all resulting subgraphs are complete. In the List Partition Problem a vertex partition has to be found and a given matrix encodes whether edges are required, forbidden or arbitrary between the partitioning subsets ([13]). The problem to partition the vertices of a graph into a minimum number of cliques was analyzed by Pirwani and Salavatipour [74]. Their work considered only a special set of graphs called unit disk graphs. Another special set of graphs (random geometric graphs) was studied by Mahjoub, Leskovskaya and Matula [65]. They engaged the problem to partition the vertices of a graph into a maximum number of independent dominating sets. In another problem derived from graph theory the goal is to find a partition that minimizes the maximum number of edges leaving the partitioning subsets ([6]). Yan and Chang [89] studied an edge partition problem for so-called block graphs. In this problem the edges of a graph have to be partitioned into a minimum number of vertex disjunct paths.
To give an example derived from computer science the Minimum Common String Partition Problem (MCSP) shall be mentioned ([19], [36]). In MCSP two given strings are partitioned into the same collection of substrings such that the number of these substrings is minimal.

All of these partition problems are of structural difficulty and do not involve a special weight function as input. An example for a partition problem involving a weight function that values elements of the basic set is the Optimum Cost Chromatic Partition Problem (OCCP) for graphs ([53], [55]). In this problem a set of different colors, which are weighted by some input function, is given. The goal is to find a coloring of the vertices such that neighboring vertices have different colors and the weight of the used colors is minimal. Feo, Goldschmidt and Khellaf [31] studied the problem to partition the vertices of an edge weighted graph. The objective in this problem is to minimize the sum of edges that have both endpoints in the same partitioning set.
A prominent example for a minimization type partition problem that involves a weight function on tuples of elements was introduced by Dantzig and Ramser [24] in 1959. It is well-known today as the Vehicle Routing Problem (VRP). In this problem one central depot in which $n$ vehicles are stationed is given. Further, there is a set of customers which have to be visited by these vehicles. The weight function, for example, is the distance or travel time between two locations. The

problem is to find a partition of the customers into $n$ subsets and $n$ tours such that each vehicle visits one subset of customers. Vehicles start the tour at the depot and return after visiting all customers. The target is to minimize the total distance or travel time of all vehicles. Due to Lenstra and Rinnooy Kan [59] VRP is $\mathbb{NP}$-complete. This directly follows from the hardness of the so-called Travelling Salesman Problem by Karp [52], which equals VRP for $n = 1$. Even for one vehicle the problem to find a shortest tour through a set of customers is $\mathbb{NP}$-complete.

Besides the standard formulation of VRP ([73]) many extensions of the problem exist. In the following an extensive list of variants is provided since the first of the partition problems studied in this work is motivated by a similar problem and thus delineation of existing work is necessary.
Very often versions of VRP are studied in which customers have a certain demand of resources. These resources have to be delivered by capacity bounded vehicles ([75]). Incompatibility constraints that restrict certain resources within one vehicle ([67]) or haul constraints that restrict travel times for vehicle drivers ([76]) are possible. Further, vehicles can be used multiple times ([4]) or split deliveries can be allowed ([28]). This means that vehicles can return to the depot before visiting further customers or the demand of a customer can also be accommodated by multiple vehicles. Golden, Assad, Levy and Gheysens [40] studied a problem setting involving nonidentical vehicles that have different cost and capacity structures. Additionally, customers can also give resources back to the vehicles ([41]) and the capacity of the vehicles can depend on the loading sequence of the resources ([72]). Gutiérrez-Jarpa, Desaulniers, Laporte and Marianov [43] considered a version of this problem where just profitable resource pickups are made. Another very popular extension of VRP are time windows for the customers ([21], [20]). In this case customers cannot be visited anytime but have to be served within certain time limits. Also soft time windows are considered ([35]). This means a penalty has to be paid if the customer is visited outside the window. The Multi-Depot Vehicle Routing Problem ([22], [45]) is also of great practical relevance. In this setting the vehicles are not located at one single depot but at multiple depots from which customers can be visited. Vidal, Crainic, Gendreau, Lahrichi and Rei [86] analyzed a periodic model in which customers have to be visited frequently within some period. In all cases also stochastic demands or travel times are imaginable ([67], [20]).

Further, dynamic customer requests or travel time changes that may happen during the routing process can be considered in solving methods ([64]). Ceschia, Di Gaspero and Schaerf [14] imposed that vehicles are owned by external carriers. These carriers bill costs to the routing company in different ways. Baldacci, Mingozzi, Roberti and Calvo [5] introduced satellite depots. Satellites are supplied from a central depot and customers are just supplied from the satellites. This leads to a two level routing problem.
Also different target functions are studied. For example, special edge sets can cost a one-time fee if accessed ([77]). This models road user charges. The target can also be to find the smallest number of vehicles which is sufficient to serve all customers ([40]) or to minimize the sum of the arrival times at the customers ([71]).

An example for a minimization type partition problem that involves a multidimensional weight function on single elements and a weight function on tuples of elements was introduced and studied by Borgwardt, Brieden and Gritzmann [8]. It is called Constrained Minimum-$k$-Star Clustering (CSC). Not only a minimal partition with respect to the target function has to be found but also multiple conditions have to be fulfilled. These conditions depend on the multidimensional weight function.
CSC evolves from the consolidation of farmland, where the problem is to reassign the land of the participating farmers. The goal is to arrange the lots of one farmer as close together as possible to reduce travel expenses. In this process a lot of constraints, like land quality or personal preferences, have to be considered. The set of all lots has to be partitioned such that every farmer gets a subset of lots, where quality or preference constraints have to be met and travel distances are minimal.
Borgwardt, Brieden and Gritzmann [8] showed that CSC is $\mathbb{NP}$-complete even for two farmers and a one-dimensional weight function on the lots. On the positive side, using the concept of total unimodularity, they showed that the problem is solvable in polynomial time if the instances are restricted to uniform weights.

Note that all examples mentioned above and also all other common partition problems just consider constant weights. This means only one constant value, which does not depend on other problem parameters, is assigned to each element or tuple. Later it is demonstrated that this is a strong limitation. In many practical applications nonconstant weight functions, which also depend on the partition itself, are necessary. But first another class of partition problems is presented.

## Partition as assignment problems

Many problems can be rephrased such that they belong to the class of partition problems. For example, assignment problems can also be understood as partition problems. In assignment problems two finite sets $A$ and $B$ are given and each element of $A$ has to be assigned to an element of $B$. Of course, in most cases some conditions and a minimization criterion have to be fulfilled. This problem also can be stated as a partition problem. Consider $A$ as the basic set which has to be partitioned into $|B|$ subsets. Thus, each subset is related to some element in $B$. Solutions of the partition problem correspond to solutions of the assignment problem in the way that all elements of one subset are assigned to the according element in $B$ and vice versa all elements assigned to an element in $B$ form a subset. Hence, partition and assignment problems just differ in the notation. Usually problems are denoted in the way in which they can be specified more easily.

A famous assignment problem is the Quadratic Assignment Problem (QAP). In QAP it is $|A| = |B|$ and an one to one assignment has to be found. Further, a quadratic target function is assumed ([56]). QAP has many applications in location theory or scheduling ([12]). For example, campaign itinerary planning for politicians is a QAP ([56]). Since QAP is $\mathbb{NP}$-complete, only small instances are solved optimally ([78]). For larger instances suboptimal algorithms are developed ([39]). Also ant systems ([66]) or genetic algorithms ([85]) are used to solve QAP.
There are many other practical problems which lead to assignment problems. Alejo, Díaz-Báñez, Cobano, Pérez-Lantero and Ollero [2] studied a velocity assignment problem for aircrafts sharing airspace to prevent collisions. Also the distribution of guests to hotel rooms to reduce gaps in the room occupancy is an assignment problem ([63]). In logistics, where goods have to be stored in special slots in warehouses, naturally assignment problems arise ([18]). The problem to assign channels to overlapping WLANs was analyzed by Elwekeil, Alghoniemy, Furukawa and Muta [30].

Another important class of assignment problems are job scheduling problems. Since the second problem studied in this work belongs to this problem class, an extensive overview of existing research is given.
In job scheduling a set of jobs and a set of machines are given. The problem is to find a processing schedule of the jobs on the machines such that the schedule is feasible or, in addition to that, some target function is minimized. Machines can only handle one job at a time. Thus, the jobs are assigned to time slots on the machines such that no two jobs are processed at the same time on the same machine. In the partition version of this problem for each machine a subset of jobs which have to be processed by the according machine must be found.
The following informal definition is taken from Applegate and Cook [3] and represents a variant in which each job has to be processed on several machines in a certain order. The target is to minimize the completion time of the last finished job.

**Definition** (Job-Shop Problem)
*The* Job-Shop Problem *(JSP) is to schedule a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time and the fact that each job has a specified processing order through the machines. The objective is to schedule the jobs so as to minimize the maximum of their completion times.*

Already in 1979 Lenstra and Rinnooy Kan [60] showed that JSP is $\mathbb{NP}$-complete. There are also a lot of other variants to the job scheduling problem. For example, jobs may have to visit some machines more often ([29]). Bartusch, Möhring and Radermacher [7] proposed a version with resource constraints. This means processing a job binds a certain amount of a resource that cannot be used by other processes. This amount is released when the job is finished and can be used for the processing of other jobs. Not only one but a whole set of different resources is possible. Additionally, the availability of the resources may change over time or the duration of processing a job may depend on the amount of available resources ([87]). Further, the starting time of processing a job may be constricted by constraints. Thus, starting a job is just allowed in a certain time window ([7]). Additionally, availability constraints for machines are possible ([58]). Setup times to prepare machines for the next job were studied by Mika, Waligora and Węglarz [68]. These setup times can also depend on the previous job on the machine ([27]). Lenstra, Shmoys and Tardos [61] introduced

a setting in which the jobs do not have to be processed by all machines but by a subset of machines. The goal is to minimize the makespan of the production schedule. A project scheduling variant was analyzed by Brucker, Drexl, Möhring, Neumann and Pesch [11]. The jobs are interpreted as projects which have temporal dependencies. Thus, certain jobs must be finished before others can be started.
Also all kinds of target functions are conceivable. For example, Lee and Kim [57] or Yang, Yang and Cheng [90] studied the Earliness-Tardiness Job Scheduling Problem, in which all jobs should be finished at a given due date. Finishing before or after this date is punished by some penalty factor. Moradi, Fatemi Ghomi and Zandieh [69] studied a job-shop problem which involves maintenance of machines. The two-sided goal in their work was to minimize the makespan of the production part and to minimize the system unavailability for the maintenance part. Maintenance can be interrupted and resumed later ([62]). Also versions with aging jobs and deteriorating maintenance of machines are considered ([90], [17], [49]). Thus, the durations of jobs and maintenance activities depend on their starting time. Mosheiov [70] studied a variant in which the deterioration of the jobs does not depend on time but on the position in the production sequence. There can also be job dependent learning effects which increase the production rate of following jobs on a machine ([51], [16], [49]). Hall and Sriskandarajah [44] or Schuster and Framinan [81] analyzed blocking or no-wait production settings in which jobs stay on the actual machine until the next machine to process the job gets free. A version that involves transportation constraints was studied by Soukhal, Oulamara and Martineau [84]. In this setting jobs or products are transported to the next machine or customer by vehicles. In this problem capacity and time of this transportation is also taken into account.

The complexity of the job scheduling variants mentioned above mainly results from the additional constraints like time windows, setup times or maintenance activities. Most of these are already studied extensively, whereas different target functions are proposed rather rarely. In most cases the objective is a linear function like the makespan or an earliness-tardiness function for due dates. Nevertheless, nonlinear or at least only piecewise linear target functions are very common in practical settings.
In the following an outsourcing problem is presented. In this problem naturally

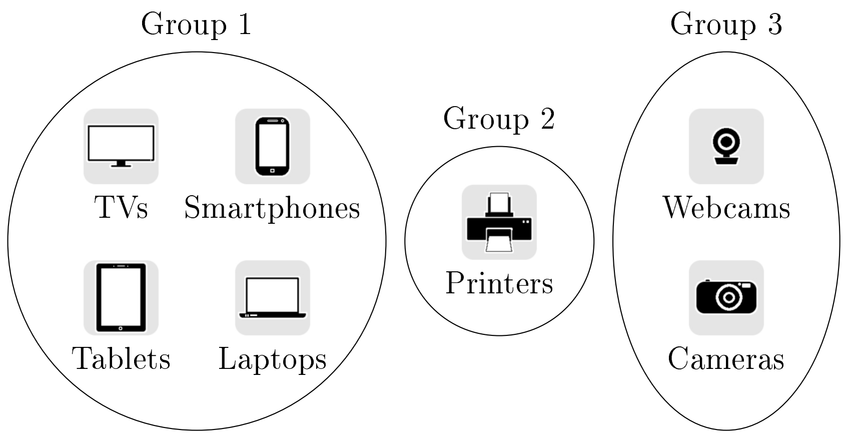

Figure 1: First partition problem in the outsourcing problem

a piecewise linear convex target function is entailed. In addition to that a non-constant weight function is required to express the inherent partition problem.

## An outsourcing problem

Consider the following situation. A big electronic enterprise produces electronic devices and all necessary electronic pieces. To focus on the production of the latter in the subsequent years the assembly of the devices will be outsourced to extern manufacturers. These can only process one order at a time. As the manufacturers have to be supplied with the relevant electronic pieces, logistics costs incur. Therefore, groups of devices that share many electronic parts are built. In this way the logistics costs can be reduced since a smaller variety of parts has to be delivered to the manufacturer. Further, each manufacturer handles only one group of devices. But one group can be outsourced to more than one manufacturer as often one manufacturer is not enough to manage all orders in that group. For example, manufacturers $M_1$ and $M_2$ both process the group „Smartphones and Tablets“. But $M_1$ builds the phone $P_1$ whereas $M_2$ builds tablet $T_1$ since the orders for $P_1$ and $T_1$ have to be processed at the same time. For a fixed price each manufacturer offers a fixed amount of working hours for the whole period. Price and working hours are negotiated beforehand. If the orders take more time than the fixed amount, an additional fee per extra hour has to be paid. This problem setting contains two different partition problems. The first is the grouping of the devices, which is obviously a partition problem.

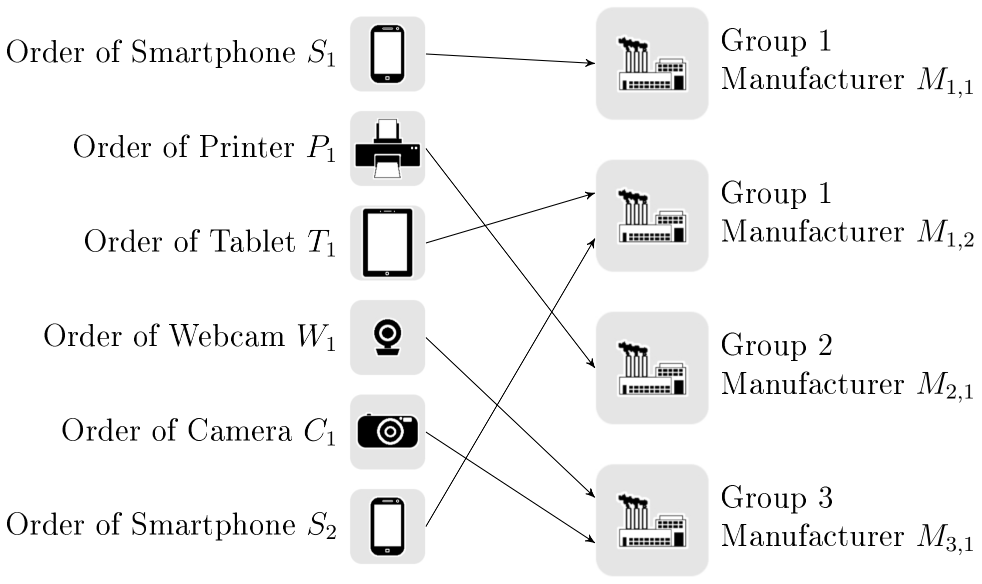
Order of Smartphone $S_1$
Order of Printer $P_1$
Order of Tablet $T_1$
Order of Webcam $W_1$
Order of Camera $C_1$
Order of Smartphone $S_2$
Group 1 Manufacturer $M_{1,1}$
Group 1 Manufacturer $M_{1,2}$
Group 2 Manufacturer $M_{2,1}$
Group 3 Manufacturer $M_{3,1}$

Figure 2: Second partition problem in the outsourcing problem

The costs of one device are composed of the production costs for the electronic pieces and the logistics costs. The production costs of the pieces are fixed for each device and are known in advance. But the logistics costs depend on the other devices in the group. If two devices share components, these have to be delivered to the manufacturer only once in a higher quantity. This is cheaper than handling a large variety of pieces. Thus, the relative logistic costs per device decrease if similar devices are outsourced together. This means the costs or, in a wider sense, the weight of a device is not constant. It also depends on the corresponding group or subset of devices.
The second partition problem is the distribution of orders to manufacturers. In terms of a partition problem, the orders have to be partitioned into subsets, where each subset corresponds to one manufacturer. Or simpler in terms of an assignment problem, each order has to be assigned to a manufacturer. Orders assigned to one manufacturer are not allowed to overlap since manufacturers can only process one order at a time. The function describing the manufacturer costs is piecewise linear and convex. Until the fixed amount of working hours is not exceeded a manufacturer just costs the constant fixed price, but after that the costs increase linearly.

Nonconstant weight functions and/or piecewise linear convex target functions also naturally appear in many other applications in which objects share common resources, like in traffic routing or in demand and supply problems, and/or where a base amount of service is included in a fixed price, like for car rentals or for mobile internet rates. Thus, studying nonconstant weight functions and piecewise linear convex target functions in combination with partition problems in a general form is of great relevance and has been neglected until now. In this work both function types are discussed separately in connection with two different partition problem classes and with different focus. Nevertheless, in both cases a strong connection to NPP, as mentioned before, exists.

At the beginning of this work a chapter of preliminaries is given. In this chapter the notation is clarified and needed previous knowledge is specified. Furthermore, some definitions and results from complexity and graph theory, which are required later, are stated.
It follows a chapter that gives a short overview of the results. The two partition problem classes introduced in this work are briefly outlined and major statements are presented.
In the third chapter partition problems with nonconstant weight functions are introduced. Therefore, a new class of weight functions, which also depend on subsets of the basic set, is defined. Special monotonic properties, motivated by practical problems like the above mentioned outsourcing problem, are defined for this class. Using these new weights a set of decision and minimization type partition problems can be introduced. The major focus in this chapter lies on the complexity analysis for those new problems. Further, the influence of the monotonic properties on the complexity is studied. Approximation algorithms are developed and proofs for the corresponding approximation ratios are given. Also heuristics are presented and tested along with the approximation algorithms on randomly generated instances.
Following a special partition problem in which jobs have to be distributed among machines of different types is proposed in the fourth chapter. In accordance with the outsourcing problem the target function is piecewise linear and convex. For an easier notation the problem is specified as an assignment problem. Initially the starting times of the jobs are assumed to be fixed, but also movable jobs are introduced. To find solutions models based on integer linear programming are developed. These models are on the major focus in this chapter. Contrary to the most practical approaches via genetic algorithms and other heuristics in this way the problem can be solved optimally. Deliberately for movable jobs

an approach is chosen in which the time is not needed to be discretized since this becomes insufficient for long periods. Rather a set of continuous variables is introduced, which are expected to be „easy“ for linear optimization. Proofs to verify the correctness of linear constraints and improvements to reduce the number of variables and constraints are essential. Especially the latter is important for the case of movable jobs since the number of variables and constraints is quadratic in the number of jobs. Finally, some heuristics are created and numerical tests of the models are performed on randomly generated instances.
In the last chapter all results are summarized and evaluated. Additionally, interesting topics for future research are pointed out.

# Chapter 1

# Preliminaries

The nomenclature used in this work is based on the common use in mathematical topics. Nevertheless, in the following a short overview of all used symbols and identifiers is given. Prerequisite background in complexity and graph theory which is assumed to be known is outlined. All needed knowledge can be found in pertinent literature.

## 1.1 Notation and wording

By default the natural numbers $\mathbb{N} := \{1, 2, \ldots\}$ are defined without 0. The natural numbers including 0 are denoted by $\mathbb{N}_0 := \{0, 1, 2, \ldots\}$.
For $a \in \mathbb{R}$ the following intervals are defined:

$$\mathbb{R}_{\geq a} := [a, \infty), \quad \mathbb{R}_{>a} := (a, \infty), \quad \mathbb{R}^+ := \mathbb{R}_{>0}, \quad \mathbb{R}_0^+ := \mathbb{R}_{\geq 0}$$

The set $\mathbb{R}^+$ is called the set of *positive* real numbers and $\mathbb{R}_0^+$ the set of *nonnegative* real numbers.
For an arbitrary set $V$ the power set is defined as

$$\mathcal{P}(V) := \{W : W \subseteq V\}.$$

For an arbitrary set $X$ and $f : X \to \mathbb{R}$ the argmin function returns the elements $Y \subseteq X$ that minimize $f$ over $X$:

$$Y = \operatorname*{argmin}_{x \in X} f(x) :\Leftrightarrow f(y) \leq f(x) \quad \forall x \in X, y \in Y$$

Likewise the argmax function is defined as follows:

$$Y = \operatorname*{argmax}_{x \in X} f(x) :\Leftrightarrow f(y) \geq f(x) \quad \forall x \in X, y \in Y$$

The following definition formally introduces the term „partition“.

**Definition 1.1.1**
*Let $V$ be an arbitrary set and $I$ be an indexing set. A family $\mathcal{W} = \{W_i\}_{i \in I}$ with $W_i \subseteq V$ is called* partition *of $V$ if and only if*

$$W_i \cap W_j = \emptyset$$

*for all $i, j \in I$ and*

$$\bigcup_{i \in I} W_i = V.$$

*The sets $W_i$ are called* partitioning sets *of $\mathcal{W}$.*

Finally, the signum function is defined as

$$\operatorname{sgn} : \mathbb{R} \to \{-1, 0, 1\},\ x \mapsto \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}.$$

At some points in this work pieces of text are connected by a vertical line. This notation is only used for structural reasons in remarks or proofs to denote an embedded proof of an embedded claim. See this in the following example theorem.

**Theorem**
*If $A$ then $C$.*

*Proof*
If $A$ then $B$:

| Arguments proving that $B$ follows from $A$.

It is known that $C$ follows from $B$. Thus, $C$ follows from $A$. $\square$

## 1.2 Complexity Theory

Complexity theory is mostly needed in Chapter 3, where the complexity of partition problems with nonconstant weight functions is analyzed. Basic knowledge can be found in Hopcraft, Motwani and Ullman [47] or Hromkovič [48]. In this regard, the reader has to be familiar with the concepts of decision and minimization type problems. The terms *feasible* and *solvable* as well as *infeasible* and *unsolvable* are used as synonyms. By „a function is computable in polynomial time“ it is meant that the function can be evaluated at every point of the domain in polynomial time and space on a Turing machine. The complexity classes $\mathbb{P}$, $\mathbb{NP}$, $\mathbb{NP}$-hard and $\mathbb{NP}$-complete are assumed to be known along with methods of proof to transfer the complexity class of one problem to another. Likewise, the big $\mathcal{O}$ notation or the term „algorithm“and associated definitions are not explained any further. An introduction can be found in Jansen and Margraf [50] or Wanka [88]. However, the terms „approximation algorithm“ and „approximation ratio“ are introduced in the following definition as in literature the denotation of these terms is sometimes not consistent.

**Definition 1.2.1**
*Let $P$ be a minimization problem and $c : X \to [1, \infty)$ a function, where $X$ is the set of possible inputs for $P$. An algorithm $\mathcal{A}$ is called* $c(x)$-approximation algorithm *if and only if for every input $x \in X$ with optimal solution value $z_x^*$ the inequality*

$$z'_x \leq c(x) \cdot z_x^*$$

*holds, where $z'_x$ is the solution value returned by $\mathcal{A}$ for input $x$. Further, $c(x)$ is called* approximation ratio *or* approximation factor.

An easy way to prove that a problem belongs to a certain complexity class is to take a problem from this class and to show that this problem can be solved by transforming it to the given problem. In the following two $\mathbb{NP}$-complete problems are specified that will be used as such representatives later.

**Definition 1.2.2** (Number Partition Problem)
*Let $S \subseteq \mathbb{N}$ be a finite set. The Problem to find a partition $\{T_1, T_2\}$ of $S$ with*

$$\sum_{s \in T_1} s = \sum_{s \in T_2} s$$

*is called* Number Partition Problem *(NPP).*

Already in 1979 Garey and Johnson [37] showed that NPP is $\mathbb{NP}$-complete. Obviously, the subset sums have to be equal to half of the sum of all elements in $S$. This is stated in the next lemma.

**Lemma 1.2.3**
*Let $\mathcal{I} = (S)$ be an instance of NPP, $\{T_1, T_2\}$ a valid partition for $\mathcal{I}$ and $\overline{S} := \sum_{s \in S} s$. Then*

$$\sum_{s \in T_1} s = \sum_{s \in T_2} s = \frac{\overline{S}}{2}.$$

*Proof*
It is

$$2 \sum_{s \in T_1} s = \sum_{s \in T_1} s + \sum_{s \in T_1} s = \sum_{s \in T_2} s + \sum_{s \in S \setminus T_2} s = \sum_{s \in S} s = \overline{S}.$$

The same argument works for $T_2$. □

Another $\mathbb{NP}$-complete problem is to find a subset of elements which sum up to a given value.

**Definition 1.2.4** (Subset Sum Problem)
*Let $S \subseteq \mathbb{N}$ be a finite set and $\sigma \in \mathbb{N}$. The Problem to find $T \subseteq S$ with*

$$\sum_{s \in T} s = \sigma$$

*is called* Subset Sum Problem *(SSP).*

Referring to Karp [52] SSP is $\mathbb{NP}$-complete.

## 1.3 Graph Theory

The basic concepts of graph theory are expected to be known. An overview of all relevant topics is given by Gritzmann [42] or Diestel [25]. In this work all graphs are assumed to be undirected, simple and loop-free if not mentioned differently. Graph theory is mainly used in Section 4.1.2 to derive an upper bound for the number of machines needed to process a set of jobs. In the following some denotations are introduced which sometimes are not consistent in literature and thus have to be made clear.

**Definition 1.3.1**
*Let $G = (V, E)$ be a graph. The* neighborhood $\delta_G(v)$ *of a vertex $v \in V$ in $G$ is defined as*

$$\delta_G(v) := \{w \in V : \{v, w\} \in E\}.$$

*A set $\mathfrak{V} \subseteq V$ is called* clique *or* $|\mathfrak{V}|$-clique *if and only if*

$$\{\{v, w\} : v, w \in \mathfrak{V}, v \neq w\} \subseteq E.$$

*A function $f : V \to \mathbb{N}$ is called* coloring *of $G$ if and only if for all $v \in V$*

$$f(v) \neq f(w)$$

*for all $w \in \delta_G(v)$. A coloring with $f(v) \leq k$ for all $v \in V$ and $k \in \mathbb{N}$ is called* $k$-coloring. *$G$ is called* $k$-colorable *if it has a $k$-coloring.*

The following definition introduces the term „chord“, which is not covered in standard literature but is needed later on.

**Definition 1.3.2**
*Let $G = (V, E)$ be a graph and $C$ a cycle in $G$. An edge $e \in E$ is called* chord *of $C$ if it connects two vertices in $C$ which are not already connected with an edge belonging to $C$.*
*A graph in which each cycle with length more than three has at least one chord is called* chordal.

The following theorem was proven by Dirac [26] in 1961 and establishes a connection between colorability and the cliques of a graph.

**Theorem 1.3.3**
*If a chordal graph is not $k$-colorable, then it contains a $(k+1)$-clique.*

*Proof*
See the proof of Theorem 3 in [26], where chordal graphs are called rigid circuit graphs. □

## 1.4 Linear programming, implementation and numerical tests

It is assumed that the reader is familiar with linear programming, in particular, with integer linear programming and the standard ways to express integer linear programs (ILPs) with linear equalities and inequalities. Basic concepts of linear programming can be found in Korte and Vygen [54], Dantzig [23], Gass [38] or Solow [82]. For integer linear programming see Schrijver [80] or Salkin [79].
All algorithms in this work are implemented in the IVE Editor [33] with Xpress-Mosel [34], which are developed by FICO [32]. Xpress-Mosel is mainly used to solve ILPs. Nevertheless, in this work also algorithms which do not involve linear optimization are implemented in Xpress-Mosel to ensure the comparability of runtimes.
The size of problem instances depends strongly on the area of application. In the outsourcing problem (see Introduction) 50 devices and 10 extern manufacturers seem to be a reasonable size, but for biological problems, in which genes have to be analyzed, the input size should be much larger. Since the problems introduced in this work are applicable in various fields, the expression „instances with a size of practical relevance“ is not applicable in this context. Thus, the test instances in this work are generated such that they can be solved by the algorithms in about 1 minute or less. Trends in runtimes are already observable in this way.
All numerical tests are implemented and performed on a 64-bit Windows system with Intel Core i7 860 CPU (2.80GHz, 4 Cores) and 4GB memory.

# Chapter 2

# Result overview

In this work two different classes of partition problems are introduced. Whereas for the first class an analysis of complexity is in the center attention, for the second class modeling the problems as ILPs is of main interest. In both cases algorithms are developed and tested afterwards. To give a brief overview all definitions and statements are presented in a concentrated way. A formally more precise and detailed presentation can be found in the corresponding chapters.

## 2.1 Nonconstant Weight Partition Problems

Nonconstant weight partition problems are introduced in a very general form. As basic set some finite set $V$ is given. This set has to be partitioned into $k \in \mathbb{N}$ subsets. For each subset a bounding value $b_i \in \mathbb{R}_0^+$ is given. Partitions of $V$ are denoted by $\mathcal{W}$. The most important part is the nonconstant weight function $p$, which not only depends on a $v \in V$ but also on a subset $W \subseteq V$. Thus, formally it is $p : \mathcal{P}(V) \times V \to \mathbb{R}$. The cumulated weights of a subset are denoted by the function $P(W) = \sum_{v \in W} p(W, v)$.
The weight function $p$ is called monotonically decreasing if for all $W' \subseteq W \subseteq V$ and all $v \in V$ it is $p(W', v) \geq p(W, v)$ and $P$ is called monotonically increasing if it is $P(W') \leq P(W)$ (Definitions 3.0.1 and 3.0.2).
With these definitions the following decision type partition problem can be formulated.

**Definition** (Definition 3.1.1)
*The problem to find a partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ with $P(W_i) \leq b_i$ for $i \in \{1, \ldots, k\}$ is called* Feasible Partition Problem *(FPP).*

Even for monotonic $p$ and $P$ this problem is $\mathbb{NP}$-complete.

**Theorem** (Theorem 3.1.3)
*FPP is $\mathbb{NP}$-complete even if $p$ is monotonically decreasing and $P$ is monotonically increasing.*

Thus, a heuristic based on integer linear programming is developed that iteratively creates partitions. The weights in the current partition are predicted by regarding the weights of the previous partition. In this way feasible solutions can be found in reasonable time, whereas for larger instances a random approach does not find solutions at all.

Also two minimization type partition problems are introduced. These involve the sum of all subset weights $s_p(\mathcal{W}) = \sum_{W \in \mathcal{W}} P(W)$ and the maximum of all subset weights $m_p(\mathcal{W}) = \max_{W \in \mathcal{W}} P(W)$ as target functions.

**Definition** (Definitions 3.2.3 and 3.2.13)
*The problem to find a partition $\mathcal{W}$ of $V$, where $s_p(\mathcal{W})$ is minimal, is called* MinSum Partition Problem *(MinSumPP).*
*The problem to find a partition $\mathcal{W}$ of $V$, where $m_p(\mathcal{W})$ is minimal, is called* MinMax Partition Problem *(MinMaxPP).*

In this general form both problems do not admit approximation algorithms.

**Theorem** (Theorems 3.2.4 and 3.2.14)
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP or MinMaxPP with approximation ratio $f$ for some function $f : \mathcal{X} \to [1, \infty)$, where $\mathcal{X}$ is the set of possible instances.*

Even if $P$ is monotonically increasing MinSumPP and MinMaxPP do not admit approximation algorithms with an exponential approximation ratio.

**Theorem** (Theorems 3.2.9 and 3.2.19)
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP or MinMaxPP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $P$ is monotonically increasing.*

But if on the other hand $p$ is assumed to be monotonically decreasing the situation changes completely. For monotonically decreasing $p$ the trivial solution (one partitioning set contains all elements of $V$) even is optimal for MinSumPP. For MinMaxPP the trivial solution is an approximation with a ratio that still depends on the values of $p$ and $|V|$. This is shown in Lemmas 3.2.5 and 3.2.15. To exclude these special cases for both problems nonempty versions are introduced.

**Definition** (Definitions 3.2.6 and 3.2.16)
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $W_i \neq \emptyset$ for $i \in \{1, \dots, k\}$, where $s_p(\mathcal{W})$ is minimal, is called* Nonempty MinSum Partition Problem *(MinSumPP$_{\neq\emptyset}$).*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $W_i \neq \emptyset$ for $i \in \{1, \dots, k\}$, where $m_p(\mathcal{W})$ is minimal, is called* Nonempty MinMax Partition Problem *(MinMaxPP$_{\neq\emptyset}$).*

If empty partitioning sets are forbidden for monotonically decreasing $p$, no approximation algorithm with an exponential approximation ratio can exist. A result like above concerning a monotonically increasing $P$ also still holds.

**Theorem** (Theorems 3.2.8 and 3.2.18)
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP$_{\neq\emptyset}$ or MinMaxPP$_{\neq\emptyset}$ with approximation ratio in $\mathcal{O}\left(2^{\mathrm{poly}(|V|,k)}\right)$ for some polynomial* $\mathrm{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $p$ is monotonically decreasing.*
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP$_{\neq\emptyset}$ or MinMaxPP$_{\neq\emptyset}$ with approximation ratio in $\mathcal{O}\left(2^{\mathrm{poly}(|V|,k)}\right)$ for some polynomial* $\mathrm{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $P$ is monotonically increasing.*

This situation again changes if both monotonic properties are considered together. In this case for MinSumPP$_{\neq\emptyset}$ as well as for MinMaxPP$_{\neq\emptyset}$ approximation algorithms can be found. The approximation ratio of the algorithm for MinSumPP$_{\neq\emptyset}$ is $(1 + |V| - k)$, for MinMaxPP$_{\neq\emptyset}$ it is $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$. Since both algorithms produce nonempty partitioning sets, they also can be used to approximate MinSumPP and MinMaxPP. This is especially important for MinMaxPP because the approximation ratio of the trivial solution still depends on the values of $p$. The according algorithms and results can be found in Section 3.2.2

Besides the two approximation algorithms also a heuristic is developed for the general cases of MinSumPP and MinMaxPP. The idea of this heuristic is to start with a partition and then to improve the target function by moving or switching elements between the partitioning sets. In the numerical tests the approximations from above are used as start partitions for the heuristic. The

results are compared to an algorithm that randomly generates solutions. The heuristic is presented in Section 3.2.1.
For MinSumPP the results of the approximation are not outstanding and equal the results of the randomized algorithm. This lies in the mediocre approximation ratio of $(1 + |V| - k)$. Nevertheless, the heuristic can improve the solution in average about 12%. Finding a better approximation algorithm is a goal in future research.
However, for MinSumPP the results are really convincing. The ratio of the approximation algorithm is much better than that for MinSumPP and this is reflected in the results. The approximation is about 39% better than the randomized algorithm and in combination with the heuristic even about 46%.

## 2.2 A partition problem with convex target function

The second partition problem in this work belongs to the class of job scheduling problems. It is denoted as an assignment problem in which each job has to be assigned to a machine. The set of machines is $M$. Each machine is of a certain type. The set of types is called $T$. In different problems either jobs $J$ or movable jobs $K$ are considered. A job is a tuple containing its starting time and the duration $d$. A movable job is a triple containing a lower and an upper bound for the starting time and also the duration. The function $\hat{t}$ assigns each machine to its type and the restriction matrix $r$ encodes whether a job or movable job can be processed by a certain machine type. For a subset of jobs $J'$ the indicator $\rho_{J'}$ represents the maximum number of jobs that have to be processed at the same time. It is shown that this number also equals the number of needed machines to process the jobs (Theorem 4.1.22). Thus, an assignment of the jobs to $\rho_{J'}$ machines exists such that no two jobs on a machine overlap.
The constants $f$, $c_0$, $c_1$ and $c_2$ are needed to model a piecewise linear convex function with one bend. This function is used as target function. The parameter $c_0$ is the factor for the constant part and models the fixed price that has to be paid for a machine if it is used. The parameters $c_1$ and $c_2$ are the two different slopes of the function and $f$ is the point where the slope changes from $c_1$ to $c_2$. To get a convex function $c_1 \leq c_2$ is presumed. The fact that only one bend is

allowed is no loss of generality since more bends can be introduced in the same way. This leads to the following problem.

**Definition** (Definition 4.1.2)
*The problem to find an assignment $a : J \to M$ with $\rho_{a^{-1}(m)} \leq 1$ for all $m \in M$ and $r_{j,\hat{t}(a(j))} = 1$ for all $j \in J$, where*

$$c_0|a(J)| + \sum_{m \in M} \max \left\{ c_1 \sum_{j \in a^{-1}(m)} d_j, c_2 \sum_{j \in a^{-1}(m)} d_j - f(c_2 - c_1) \right\}$$

*is minimal, is called* Convex Job Assignment Problem *(JAP).*

JAP does not admit an approximation algorithm with exponential approximation ratio.

**Theorem** (Theorem 4.1.3)
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves JAP with approximation ratio in $\mathcal{O}\left(2^{\mathrm{poly}(|J|,|M|,|T|,f)}\right)$ for some polynomial* $\mathrm{poly} : \mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 1}$.

To solve JAP an ILP is developed. This program can be found in Definition 4.1.4.

A similar problem can be formulated for movable jobs. Thus, not only an assignment to machines has to be found but also the starting times, which have to lie within the given bounds, have to be determined. A set of movable jobs with fixed starting times is called a corresponding set of jobs for the set of movable jobs. The problem is defined as follows.

**Definition 2.2.1** (Definition 4.1.5)
*The problem to find a feasible set of corresponding jobs $J = \{j_k\}_{k \in K}$ for $K$ and an assignment $a : J \to M$ with $\rho_{a^{-1}(m)} \leq 1$ for all $m \in M$ and $r_{k,\hat{t}(a(j_k))} = 1$ for all $k \in K$, where*

$$c_0|a(J)| + \sum_{m \in M} \max \left\{ c_1 \sum_{j \in a^{-1}(m)} d_j, c_2 \sum_{j \in a^{-1}(m)} d_j - f(c_2 - c_1) \right\}$$

*is minimal, is called* Convex Movable Job Assignment Problem *(MJAP).*

Like JAP also MJAP does not admit an approximation algorithm with exponential approximation ratio. To solve MJAP again an ILP is developed. In this process different variables are introduced and connected via logical expressions. These expressions are modeled as linear inequalities, whose correctness is proven. Further, the formulation of the program is tightened so that less variables and inequalities are needed. The resulting program can be found in Definition 4.1.16.

Besides the linear program to solve MJAP optimally also a heuristic is presented. The idea is to separate the assignment of starting times and the assignment of jobs to machines. After determining the starting times simply JPA can be solved. Since each machine costs a fixed price, the jobs are arranged in such a way that as less as possible machines are needed. As mentioned before the number of needed machines is in one to one correlation with the maximum number of jobs that overlap at a time. Thus, the jobs are arranged so that they overlap as less as possible. The ILP to solve this problem is presented in Definition 4.1.36. In the development of this problem not only the correctness of all linear constraints is proven but also an extensive analysis is done that provides criteria to reduce variables and constraints in certain cases.

Another type of heuristic for both problems, JAP and MJAP, is proposed in Section 4.2. In practical problems often the number of machines is much larger than the number of different machine types. Therefore, assigning jobs only to machine types reduces the complexity of the previously mentioned ILPs drastically. The assignment of jobs to single machines can be done in a post processing step for each machine type. The according job to machine type assignment problems for JAP and MJAP are characterized as follows.

**Definition** (Definitions 4.2.1 and 4.2.4)
*The problem to find an assignment $a : J \to T$ with $r_{j,a(j)} = 1$ for all $j \in J$ and $\rho_{a^{-1}(t)} \leq |\hat{t}^{-1}(t)|$ for all $t \in T$, where*

$$c_0 \sum_{t \in T} \rho_{a^{-1}(t)} + \sum_{t \in T} \max \left\{ c_1 \sum_{j \in a^{-1}(t)} d_j, c_2 \sum_{j \in a^{-1}(t)} d_j - \rho_{a^{-1}(t)} f(c_2 - c_1) \right\}$$

*is minimal, is called* Compact Convex Job Assignment Problem *(CJAP).*

*The problem to find a feasible set of corresponding jobs* $J = \{j_k\}_{k \in K}$ *for* $K$ *and an assignment* $a : J \to T$ *with* $r_{k,a(j_k)} = 1$ *for all* $k \in K$ *and* $\rho_{a^{-1}(t)} \leq |\hat{t}^{-1}(t)|$ *for all* $t \in T$*, where*

$$c_0 \sum_{t \in T} \rho_{a^{-1}(t)} + \sum_{t \in T} \max \left\{ c_1 \sum_{j \in a^{-1}(t)} d_j, c_2 \sum_{j \in a^{-1}(t)} d_j - \rho_{a^{-1}(t)} f(c_2 - c_1) \right\}$$

*is minimal, is called the* Compact Convex Movable Job Assignment Problem *(CMJAP).*

The corresponding ILPs can be found in Definitions 4.2.3 and 4.2.7. The assignment of jobs to single machines is also modeled as ILP. Since in practical settings often an equal workload of the machines is desirable, the job durations are evenly distributed among the machines. The according ILPs are stated in Definitions 4.2.8 and 4.2.9.

Altogether, this results in one alternative way to solve JAP and three alternative ways to solve MJAP.

- JAP:
  - (i) Solve CJAP → assign jobs to machines for each machine type
- MJAP:
  - (ii) Solve CMJAP → assign jobs to machines for each machine type
  - (iii) Heuristically determine starting times of jobs → solve JAP
  - (iv) Heuristically determine starting times of jobs → solve CJAP → assign jobs to machines for each machine type

All solution approaches are tested on randomly generated instances and compared with an algorithm that randomly tries to find feasible assignments. It turns out that all heuristics perform outstandingly and find the optimal solution for all instances. Only on specially designed instances that involve certain job structures the heuristics fail. Solving the larger instances via the primal ILP, for JAP and MJAP, takes up to one minute, whereas the heuristics (i) and (iv) run in a fraction of a second. Furthermore, the random algorithm does not find a solution for about half of the instances. This on the one hand highlights the complexity of the instances and on the other hand underlines the predominance of the heuristics.

# Chapter 3

# Nonconstant Weight Partition Problems

In all prominent partition problems constant weight functions are used. This means each element of the basic set or objects in the underlying constraints are weighted by a constant value. These elements and objects can, for example, be vertices ([8]) or edges ([24]) of a graph, natural numbers ([37]), which are simply weighted by their value, or colors ([53], [55]). In this chapter the weight functions are extended such that the weight may depend on the partition itself. Like this, synergy effects of the elements in one partitioning set can be modeled. The main results of this chapter will also be published in [10].
To introduce partition problems with nonconstant weight functions first these weight functions have to be defined. This is done in the following. Further, some monotonic properties are specified.

**Definition 3.0.1** (Subset function)
*For a finite set $V$ a function*

$$p : \mathcal{P}(V) \times V \to \mathbb{R}$$

*that is computable in polynomial time depending on the size of $V$ is called* subset function *on $V$, and* nonnegative subset function *or* positive subset function *if and only if*

$$p : \mathcal{P}(V) \times V \to \mathbb{R}_0^+$$

*or*

$$p : \mathcal{P}(V) \times V \to \mathbb{R}^+.$$

*A subset function $p$ is called* monotonically decreasing *if and only if for all $W' \subseteq W \subseteq V$ and all $v \in V$*

$$p(W', v) \geq p(W, v)$$

*holds.*

Subset functions have two parameters, a subset $W$ and an element $v \in V$. In this way weights can be assigned to $v$ that also depend on $W$. In other words, for each subset a different weight can be assigned to an element. To motivate the monotonic property the outsourcing example from the introduction is used. For each device exist a fixed absolute cost component and a relative cost component which depends on the other devices in the group. The relative costs should decrease if additional devices are added to the group since these probably have similar electronic pieces and thus relative logistics costs decrease. Exactly this property is represented by a monotonically decreasing subset function.

To weight a subset of $V$ the weights of all elements with respect to the subset are added. This leads to the following definition of cumulated subset functions. Again a monotonic property is introduced.

**Definition 3.0.2** (Cumulated subset function)
*Let $V$ be a finite set and $p$ a subset function on $V$. Then*

$$P : \mathcal{P}(V) \to \mathbb{R},\ W \mapsto \sum_{v \in W} p(W, v)$$

*is called* cumulated subset function *for $p$. For $W \subseteq V$ the value $P(W)$ is called* subset value *of $W$.*
*A cumulated subset function $P$ is called* monotonically increasing *if and only if for all $\emptyset \neq W' \subseteq W \subseteq V$*

$$P(W') \leq P(W)$$

*holds.*

**Remark 3.0.3**
To avoid additional indices or other identifiers subset functions always are denoted by lower case characters and the corresponding cumulated subset func-

tions are specified by the according upper case characters and vice versa (for example: $p$ corresponds to $P$ or $\hat{p}$ corresponds to $\hat{P}$).

The purpose of the monotonic ascent for cumulated subset functions can again be motivated by the outsourcing problem. The relative costs of the devices in a group may decrease if other devices are added. Nevertheless, the total costs for producing and delivering the electronic pieces in the bigger group should be larger. This behavior is modeled by monotonically increasing cumulated subset functions. In the definition the empty set is excluded from the subsets since $P(\emptyset) = 0$ and thus subsets could not be weighted with negative values. However, for nonnegative subset functions, which are mainly studied in this work, this is no constriction.

The following lemma or rather the subsequent corollary is used in proofs later on. It states that for nonnegative and monotonically decreasing $p$ the subset value of the union of two subsets is bounded by the sum of the subset values of these sets.

**Lemma 3.0.4**
*Let $V$ be a finite set and $p$ a nonnegative and monotonically decreasing subset function on $V$. Then for any $W, W' \subseteq V$ it is*

$$P(W \cup W') \leq P(W) + P(W').$$

*Proof*
It is

$$\begin{aligned}
P(W \cup W') &= \sum_{w \in W \cup W'} p(W \cup W', w) \\
&\leq \sum_{w \in W} p(W \cup W', w) + \sum_{w \in W'} p(W \cup W', w) \\
&\leq \sum_{w \in W} p(W, w) + \sum_{w \in W'} p(W', w) \\
&= P(W) + P(W'),
\end{aligned}$$

where the first „$\leq$“ is no „$=$“ because $W$ and $W'$ do not have to be disjunct. □

This corollary follows immediately.

**Corollary 3.0.5**
*Let $V$ be a finite set and $p$ a nonnegative and monotonically decreasing subset function on $V$. Then for any $W \subseteq V$ and $v \in V$ it is*

$$P(W \cup \{v\}) - P(W) \leq P(\{v\}).$$

The next definition describes the restriction of subset functions from $V$ to subsets of $V$. This is useful later, especially in Section 3.2.2 for proofs by induction on the size of $V$.

**Definition 3.0.6**
*Let $V$ be a finite set and $p$ a subset function on $V$. For $V' \subseteq V$ define the subset function $p_{|_{V'}} : \mathcal{P}(V') \times V' \to \mathbb{R}$ as*

$$p_{|_{V'}}(W', v') := p(W', v')$$

*for $W' \subseteq V'$ and $v' \in V'$.*

**Remark 3.0.7**
To make the notation easier and due to the canonical transformation in the following $p_{|_{V'}}$ is abbreviated as $p$. Thus, $p$ is also used as subset function for $V'$.

## 3.1 Feasible Partition Problems

In this section a decision type partition problem and its complexity is studied. The problem is to find a partition of a basic set for which the subset values are bounded by given constants.

**Definition 3.1.1** (Feasible Partition Problem)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$ and $b_i \in \mathbb{R}_0^+$ for $i \in \{1, \ldots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ with $P(W_i) \leq b_i$ for $i \in \{1, \ldots, k\}$ is called* Feasible Partition Problem *(FPP).*

NPP can be transformed to FPP. Since NPP is $\mathbb{NP}$-complete ([37]), so is FPP. This is true even if $p$ is just allowed to take the values 0 and 1 and $V$ has to be partitioned into two subsets only.

**Theorem 3.1.2**
*FPP is $\mathbb{NP}$-complete even if $p$ is $\{0,1\}$-valued and $k=2$.*

*Proof*
Let $\mathcal{I} = (S)$ be an instance of NPP. Define $\overline{S} := \sum_{s\in S} s$ and an instance $\mathcal{J} := (V,p,k,b)$ of FPP with $V := S$ and

$$p(W,v) := \begin{cases} 0 & \text{if } \sum_{w\in W} w = \frac{\overline{S}}{2} \\ 1 & \text{otherwise} \end{cases},$$

where $W \subseteq V$ and $v \in V$. Obviously, $p$ is polynomially computable in the size of $V$. Note that not every possible value of $p$ is computed at this point and just a formal definition of $p$ is given. This makes the transformation polynomial. Further, let $k := 2$ and $b_1 := b_2 := 0$.
The instance $\mathcal{J}$ is solvable if and only if $\mathcal{I}$ is solvable:

„$\Rightarrow$“: Let $\mathcal{W} = \{W_1, W_2\}$ be a feasible solution for $\mathcal{J}$ and without loss of generality $W_1 \neq \emptyset$ (because without loss of generality $V \neq \emptyset$). Since

$$P(W_1) = \sum_{v\in W_1} p(W_1,v) \le b_1 = 0$$

and $p$ is nonnegative, it follows $p(W_1,v)=0$ for all $v \in W_1$. Let $v \in W_1$. It follows $p(W_1,v)=0$ and thus

$$\sum_{w\in W_1} w = \frac{\overline{S}}{2}.$$

Further,

$$\sum_{w\in W_2} w = \sum_{v\in V} v - \sum_{w\in W_1} w = \sum_{s\in S} s - \frac{\overline{S}}{2} = \frac{\overline{S}}{2} = \sum_{w\in W_1} w.$$

This means $\{T_1, T_2\}$ with $T_1 := W_1$ and $T_2 := W_2$ is a solution for $\mathcal{I}$.

„$\Leftarrow$“: Let $\{T_1, T_2\}$ be a partition of $S$ with

$$\sum_{s\in T_1} s = \sum_{s\in T_2} s.$$

Define a partition $\mathcal{W} = \{W_1, W_2\}$ of $V$ with $W_1 := T_1$ and $W_2 := T_2$. With Lemma 1.2.3 it is

$$\sum_{w \in W_1} w = \sum_{w \in W_2} s = \frac{\overline{S}}{2}.$$

This means $p(W_1, v) = p(W_2, v) = 0$ for all $v \in V$. Therefore,

$$P(W_1) = 0 = b_1,$$
$$P(W_2) = 0 = b_2$$

and $\mathcal{W}$ is a feasible solution for $\mathcal{J}$.

Thus, FFP is $\mathbb{NP}$-hard. Furthermore, $P$ is computable in polynomial time and thus checking whether $P(W_i) \leq b_i$ for all $i \in \{1, \ldots, k\}$ is possible in polynomial time. Checking whether $\mathcal{W}$ is a partition of $V$ is possible in polynomial time, too. This means FPP $\in \mathbb{NP}$ and therefore FPP is $\mathbb{NP}$-complete. Thus, FFP is $\mathbb{NP}$-hard. Furthermore, $P$ is computable in polynomial time and thus checking whether $P(W_i) \leq b_i$ for all $i \in \{1, \ldots, k\}$ is possible in polynomial time. Checking whether $\mathcal{W}$ is a partition of $V$ is possible in polynomial time, too. This means FPP $\in \mathbb{NP}$ and therefore FPP is $\mathbb{NP}$-complete. □

The question is what happens if $p$ and $P$ satisfy the monotonic properties and in addition $p$ is positive. The next theorem states that even in this case FPP is $\mathbb{NP}$-complete. This can also simply be seen by defining $p(W, v) := v$ and choosing

$$b_1 := b_2 := \frac{\sum_{s \in S} s}{2}$$

in the last proof. Since $p$ is constant for each $v$, it is also monotonically decreasing and since $v \in \mathbb{N}$ the cumulated subset function $P$ is monotonically increasing. Nevertheless, in the next proof a different transformation with a „real“ nonconstant weight function is used that is even strictly monotonic. This means for $W' \subsetneq W \subseteq V$ and $v \in V$ it is

$$p(W', v) > p(W, v) \quad \text{and} \quad P(W') < P(W).$$

For the transformation SSP is used, which is $\mathbb{NP}$-complete ([52]).

**Theorem 3.1.3**
*FPP is $\mathbb{NP}$-complete even if $k = 2$, $p$ is positive, (strictly) monotonically decreasing and $P$ is (strictly) monotonically increasing.*

*Proof*
SSP can be polynomially transformed to FPP, where $p$ is positive, (strictly) monotonically decreasing and $P$ is (strictly) monotonically increasing.
For a given instance $\mathcal{I} = (S, \sigma)$ of SSP set $\overline{S} := \sum_{s\in S} s$ and $s_{\max} := \max_{s\in S} s$. Without loss of generality let $0 < \sigma < \overline{S}$, $|S| \geq 1$ and $s_{\max} > 1$, otherwise solving $\mathcal{I}$ would be trivial. Define an instance $\mathcal{J} := (V, p, k, b)$ of FPP with $V := S$ and

$$p(W, v) := \begin{cases} v + \displaystyle\sum_{w\in V\setminus W} \frac{w}{|W| \cdot s_{\max}} & \text{if } W \neq \emptyset \\ s_{\max} + |V| & \text{otherwise} \end{cases},$$

where $W \subseteq V$ and $v \in V$. Obviously, $p$ is polynomially computable in the size of $V$. Further, let $k := 2$,

$$b_1 := \frac{\overline{S} + (s_{\max} - 1)\sigma}{s_{\max}} > 0$$

and

$$b_2 := \frac{\overline{S} \cdot s_{\max} - (s_{\max} - 1)\sigma}{s_{\max}} = \underbrace{\overline{S} - \sigma}_{>0} + \frac{\sigma}{s_{\max}} > 0.$$

The subset function $p$ is positive since

$$\underbrace{v + \sum_{w\in V\setminus W} \frac{w}{|W| \cdot s_{\max}}}_{\geq 0} > 0$$

for $W \neq \emptyset$ and $s_{\max} + |V| > 0$. For all $\emptyset \neq W \subseteq V$ it is

$$\begin{aligned} P(W) &= \sum_{v\in W} p(W, v) \\ &= \sum_{v\in W} \left( v + \sum_{w\in V\setminus W} \frac{w}{|W| \cdot s_{\max}} \right) \end{aligned}$$

$$\begin{aligned}
&= \sum_{v\in W} v + |W| \sum_{w\in V\setminus W} \frac{w}{|W|\cdot s_{\max}} \\
&= \sum_{w\in W} w + \sum_{w\in V\setminus W} \frac{w}{s_{\max}} \\
&= \frac{s_{\max}\sum_{w\in W} w}{s_{\max}} + \frac{\overline{S} - \sum_{w\in W} w}{s_{\max}} \\
&= \frac{\overline{S} + (s_{\max}-1)\sum_{w\in W} w}{s_{\max}}.
\end{aligned}$$

Let $\emptyset \neq W' \subsetneq W \subseteq V$ and $v \in V$. $p$ is strictly monotonically decreasing because

$$\begin{aligned}
p(W', v) &= v + \sum_{w\in V\setminus W'} \frac{w}{|W'|\cdot s_{\max}} \\
&> v + \sum_{w\in V\setminus W} \frac{w}{|W'|\cdot s_{\max}} \\
&> v + \sum_{w\in V\setminus W} \frac{w}{|W|\cdot s_{\max}} \\
&= p(W, v)
\end{aligned}$$

and

$$p(\emptyset, v) = s_{\max} + |V| \geq v + \sum_{w\in V} \frac{w}{s_{\max}} > v + \sum_{w\in V\setminus W'} \frac{w}{|W'|\cdot s_{\max}}.$$

Further,

$$P(W') = \frac{\overline{S} + (s_{\max}-1)\sum_{w\in W'} w}{s_{\max}} < \frac{\overline{S} + (s_{\max}-1)\sum_{w\in W} w}{s_{\max}} = P(W)$$

holds and thus $P$ is strictly monotonically increasing.
The instance $\mathcal{J}$ is solvable if and only if $\mathcal{I}$ is solvable:

„$\Rightarrow$“: Let $\mathcal{W} = \{W_1, W_2\}$ be a feasible solution for $\mathcal{I}$. It is $W_i \neq \emptyset$ for $i \in \{1,2\}$. If for example $W_2 = \emptyset$, then $W_1 = S \neq \emptyset$ would follow and thus

$$\begin{aligned}
P(W_1) \leq b_1 &\Leftrightarrow \frac{\overline{S} + (s_{\max}-1)\sum_{w\in W_1} w}{s_{\max}} \leq \frac{\overline{S} + (s_{\max}-1)\sigma}{s_{\max}} \\
&\Leftrightarrow \frac{\overline{S} + (s_{\max}-1)\overline{S}}{s_{\max}} \leq \frac{\overline{S} + (s_{\max}-1)\sigma}{s_{\max}} \\
&\Leftrightarrow \overline{S} \leq \sigma.
\end{aligned}$$

This is in contradiction to the assumption $\sigma < \overline{S}$.
Because

$$P(W_1) = \frac{\overline{S} + (s_{\max} - 1)\sum_{w \in W_1} w}{s_{\max}} \leq b_1 = \frac{\overline{S} + (s_{\max} - 1)\sigma}{s_{\max}}$$

and

$$P(W_2) = \frac{\overline{S} + (s_{\max} - 1)\sum_{w \in W_2} w}{s_{\max}} \leq b_2 = \frac{\overline{S} \cdot s_{\max} - (s_{\max} - 1)\sigma}{s_{\max}}$$

it holds

$$\sum_{w \in W_1} w \leq \sigma$$

and

$$\begin{aligned}
&\frac{\overline{S} + (s_{\max} - 1)\sum_{w \in W_2} w}{s_{\max}} \leq \frac{\overline{S} \cdot s_{\max} - (s_{\max} - 1)\sigma}{s_{\max}} \\
\Leftrightarrow\ &\overline{S} + (s_{\max} - 1)\sum_{w \in W_2} w \leq \overline{S} \cdot s_{\max} - (s_{\max} - 1)\sigma \\
\Leftrightarrow\ &(s_{\max} - 1)\left(\sum_{w \in W_2} w + \sigma\right) \leq (s_{\max} - 1)\overline{S} \\
\Leftrightarrow\ &\sum_{w \in W_2} w + \sigma \leq \overline{S} \\
\Leftrightarrow\ &\sigma \leq \overline{S} - \sum_{w \in W_2} w.
\end{aligned}$$

Since $\mathcal{W}$ is a partition, it follows

$$\sigma \leq \overline{S} - \sum_{w \in W_2} w = \sum_{w \in W_1} w \leq \sigma$$

and $T := W_1 \subseteq S$ is valid subset for $\mathcal{I}$.

„$\Leftarrow$": Let $T \subseteq S$ with

$$\sum_{s \in T} s = \sigma.$$

Define a partition $\mathcal{W} = \{W_1, W_2\}$ of $V$ with $W_1 := T$ and $W_2 := V \backslash T$. Since $0 < \sigma < \overline{S}$, it follows $W_i \neq \emptyset$ for $i \in \{1, 2\}$. Thus,

$$\begin{aligned}
P(W_1) &= \frac{\overline{S} + (s_{\max} - 1) \sum_{w \in W_1} w}{s_{\max}} \\
&= \frac{\overline{S} + (s_{\max} - 1) \sum_{s \in T} s}{s_{\max}} \\
&= \frac{\overline{S} + (s_{\max} - 1)\sigma}{s_{\max}} \\
&= b_1
\end{aligned}$$

and

$$\begin{aligned}
P(W_2) &= \frac{\overline{S} + (s_{\max} - 1) \sum_{w \in W_2} w}{s_{\max}} \\
&= \frac{\overline{S} + (s_{\max} - 1) \left(\overline{S} - \sum_{w \in W_1} w\right)}{s_{\max}} \\
&= \frac{\overline{S} + (s_{\max} - 1) \left(\overline{S} - \sigma\right)}{s_{\max}} \\
&= \frac{\overline{S} \cdot s_{\max} - (s_{\max} - 1)\sigma}{s_{\max}} \\
&= b_2
\end{aligned}$$

and $\mathcal{W}$ is a feasible solution for $\mathcal{J}$.

This makes FFP $\mathbb{NP}$-hard. The completeness follows like before. $\square$

In many practical problems there are fixed centers which must be contained in certain partitioning sets. For example in CSC if some farmers want to keep certain lots or in the outsourcing problem if some devices have to be contained in certain groups. Therefore, FPP is extended as follows.

**Definition 3.1.4** (Feasible Partition Problem with fixed centers)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$, $b_i \in \mathbb{R}_0^+$ and $w_i \in V$ for $i \in \{1, \dots, k\}$ with $w_i \neq w_j$ for all $i, j \in \{1, \dots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $w_i \in W_i$ and $P(W_i) \leq b_i$ for $i \in \{1, \dots, k\}$ is called* Feasible Partition Problem with fixed centers *(FPPc).*

Analog complexity results like above hold also for FPPc since FPP can be transformed to FPPc.

**Theorem 3.1.5**
*FPPc is $\mathbb{NP}$-complete even if $p$ is $\{0,1\}$-valued and $k = 2$.*

*Proof*
FPP is $\mathbb{NP}$-complete even if $p$ is $\{0,1\}$-valued, $k = 2$ and $b_i = 0$ for all $i \in \{1,\dots,k\}$ (Theorem 3.1.2).
For a given instance $\mathcal{I} = (V, p, k, b)$ of FPP with an arbitrary set $V$, a $\{0,1\}$-valued nonnegative subset function $p$ on $V$, $k = 2$ and $b_1 = b_2 = 0$ define an instance $\hat{\mathcal{I}} := (\hat{V}, \hat{p}, \hat{k}, \hat{b}, \hat{w})$ of FPPc with $\hat{V} := V \cup \{\hat{w}_1, \hat{w}_2\}$, where $\hat{w}_1, \hat{w}_2 \notin V$ are two arbitrary elements with $\hat{w}_1 \neq \hat{w}_2$, and

$$\hat{p}(\hat{W}, \hat{v}) := \begin{cases} p(\hat{W} \cap V, \hat{v}) & \text{if } \hat{v} \in V \\ 0 & \text{otherwise} \end{cases},$$

where $\hat{W} \subseteq \hat{V}$ and $\hat{v} \in \hat{V}$. The subset function $\hat{p}$ is polynomially computable in the size of $\hat{V}$ because $p$ is polynomially computable. Further, let $\hat{k} := k$ and $\hat{b}_i := b_i = 0$ for $i \in \{1,2\}$. By definition $\hat{p}$ is $\{0,1\}$-valued.
The instance $\hat{\mathcal{I}}$ is solvable if and only if $\mathcal{I}$ is solvable:

„$\Rightarrow$“: Let $\hat{\mathcal{W}} = \{\hat{W}_1, \hat{W}_2\}$ be a feasible solution for $\hat{\mathcal{I}}$. The family $\mathcal{W} := \{W_1, W_2\}$ with $W_i := \hat{W}_i \cap V = \hat{W}_i \backslash \{\hat{w}_i\}$ for $i \in \{1,2\}$ is a feasible solution for $\mathcal{I}$ because

$$\begin{aligned} \hat{P}(\hat{W}_i) &= \sum_{\hat{v} \in \hat{W}_i} \hat{p}(\hat{W}_i, \hat{v}) \\ &= \sum_{\hat{v} \in W_i} \hat{p}(\hat{W}_i, \hat{v}) + \hat{p}(\hat{W}_i, \hat{w}_i) \\ &= \sum_{\hat{v} \in W_i} p(\hat{W}_i \cap V, \hat{v}) + 0 \\ &= \sum_{v \in W_i} p(W_i, v) \\ &= P(W_i) \end{aligned}$$

and thus

$$\begin{aligned} &\hat{P}(\hat{W}_i) \leq \hat{b}_i = 0 \\ \Leftrightarrow\ &P(W_i) \leq b_i = 0 \end{aligned}$$

for $i \in \{1, 2\}$.

„$\Leftarrow$“: For a feasible partition $\mathcal{W} := \{W_1, W_2\}$ for $\mathcal{I}$ define $\hat{\mathcal{W}} := \{\hat{W}_1, \hat{W}_2\}$ with $\hat{W}_i := W_i \cup \{w_i\}$ for $i \in \{1, 2\}$. Like in the „$\Rightarrow$“ part $\hat{\mathcal{W}}$ is a solution for $\hat{\mathcal{I}}$.

Thus, FFPc is $\mathbb{NP}$-hard. Checking whether $\hat{w}_i \in \hat{W}_i$ for all $i \in \{1, \dots, k\}$ is possible in polynomial time. Therefore, FPPc $\in \mathbb{NP}$ and FPPc is $\mathbb{NP}$-complete.

$\square$

The transformation in the proof of the next theorem is very similar to the one in the last proof. However, since $\hat{p}$ has to be positive an $\varepsilon \in \mathbb{R}^+$ has to be added. The main work is to verify the monotonicity. The rest follows analogously.

**Theorem 3.1.6**

*FPPc is $\mathbb{NP}$-complete even if $k = 2$ and $p$ is positive, monotonically decreasing and $P$ is monotonically increasing.*

*Proof*

FPP is $\mathbb{NP}$-hard even if $k = 2$ and $p$ is positive, monotonically decreasing and $P$ is monotonically increasing (Theorem 3.1.3).

For a given instance $\mathcal{I} = (V, p, k, b)$ of FPP with an arbitrary set $V$, $k = 2$ and a positive, monotonically decreasing subset function $p$ and monotonically increasing cumulated subset function $P$ define an instance $\hat{\mathcal{I}} := (\hat{V}, \hat{p}, \hat{k}, \hat{b}, \hat{w})$ of FPPc with $\hat{V} := V \cup \{\hat{w}_1, \hat{w}_2\}$, where $\hat{w}_1, \hat{w}_2 \notin V$ are two arbitrary elements with $\hat{w}_1 \neq \hat{w}_2$, and

$$\hat{p}(\hat{W}, \hat{v}) := \begin{cases} p(\hat{W} \cap V, \hat{v}) & \text{if } \hat{v} \in V \\ \varepsilon & \text{otherwise} \end{cases},$$

where $\hat{W} \subseteq \hat{V}$, $\hat{v} \in \hat{V}$ and $\varepsilon > 0$. The subset function $\hat{p}$ is polynomially computable in the size of $\hat{V}$ because $p$ is polynomially computable. Further, let $\hat{k} := k$ and $\hat{b}_i := b_i + \varepsilon$ for $i \in \{1, 2\}$.

The subset function $\hat{p}$ is positive since $p$ is positive. Further, $\hat{p}$ is monotonically decreasing:

Let $\hat{W}' \subseteq \hat{W} \subset \hat{V}$ and $\hat{v} \in \hat{V}$. It follows $\hat{W}' \cap V \subseteq \hat{W} \cap V$ and thus with the monotonicity of $p$ it is

$$\hat{p}(\hat{W}', \hat{v}) = p(\hat{W}' \cap V, \hat{v}) \geq p(\hat{W} \cap V, \hat{v}) = \hat{p}(\hat{W}, \hat{v})$$

for $\hat{v} \in V$ and

$$\hat{p}(\hat{W}', \hat{v}) = \varepsilon = \hat{p}(\hat{W}, \hat{v})$$

for $\hat{v} \in \hat{V} \backslash V$.

The cumulated subset function $\hat{P}$ is monotonically increasing:

For $\emptyset \neq \hat{W}' \subseteq \hat{W} \subset \hat{V}$ define $W' := \hat{W}' \cap V = \hat{W}' \backslash \{\hat{w}_1, \hat{w}_2\}$ and $W := \hat{W} \cap V = \hat{W} \backslash \{\hat{w}_1, \hat{w}_2\}$. With

$$\begin{aligned} P(W) &= \sum_{v \in W} p(W, v) \\ &= \sum_{v \in W} p(\hat{W} \cap V, v) \\ &= \sum_{v \in W} \hat{p}(\hat{W}, v) \\ &= \sum_{\hat{v} \in \hat{W} \backslash \{\hat{w}_1, \hat{w}_2\}} \hat{p}(\hat{W}, \hat{v}) \end{aligned}$$

it follows

$$\begin{aligned} \hat{P}(\hat{W}) &= \sum_{\hat{v} \in \hat{W} \backslash \{\hat{w}_1, \hat{w}_2\}} \hat{p}(\hat{W}, \hat{v}) + \sum_{\hat{v} \in \hat{W} \cap \{\hat{w}_1, \hat{w}_2\}} \hat{p}(\hat{W}, \hat{v}) \\ &= P(W) + \sum_{\hat{v} \in \hat{W} \cap \{w_1, w_2\}} \varepsilon \\ &= P(W) + |\hat{W} \cap \{w_1, w_2\}| \varepsilon \end{aligned}$$

and analogously

$$P(W') = P(W') + |\hat{W}' \cap \{w_1, w_2\}| \varepsilon.$$

Therefore,

$$\hat{P}(\hat{W}') = P(W') + |\hat{W}' \cap \{w_1, w_2\}|\varepsilon \le P(W) + |\hat{W} \cap \{w_1, w_2\}|\varepsilon = \hat{P}(\hat{W})$$

holds.

The instance $\hat{\mathcal{I}}$ is solvable if and only if $\mathcal{I}$ is solvable:

„$\Rightarrow$“: Let $\hat{\mathcal{W}} = \{\hat{W}_1, \hat{W}_2\}$ be a feasible solution for $\hat{\mathcal{I}}$. The family $\mathcal{W} := \{W_1, W_2\}$ with $W_i := \hat{W}_i \cap V = \hat{W}_i \backslash \{\hat{w}_i\}$ for $i \in \{1, 2\}$ is a feasible solution for $\mathcal{I}$ because

$$\begin{aligned}
\hat{P}(\hat{W}_i) &= \sum_{\hat{v} \in \hat{W}_i} \hat{p}(\hat{W}_i, \hat{v}) \\
&= \sum_{\hat{v} \in W_i} \hat{p}(\hat{W}_i, \hat{v}) + \hat{p}(\hat{W}_i, \hat{w}_i) \\
&= \sum_{\hat{v} \in W_i} p(\hat{W}_i \cap V, \hat{v}) + \varepsilon \\
&= \sum_{v \in W_i} p(W_i, v) + \varepsilon \\
&= P(W_i) + \varepsilon
\end{aligned}$$

and thus

$$\begin{aligned}
& \hat{P}(\hat{W}_i) \le \hat{b}_i \\
\Leftrightarrow\ & P(W_i) + \varepsilon \le b_i + \varepsilon \\
\Leftrightarrow\ & P(W_i) \le b_i
\end{aligned}$$

for $i \in \{1, 2\}$.

„$\Leftarrow$“: For a feasible partition $\mathcal{W} := \{W_1, W_2\}$ for $\mathcal{I}$ define $\hat{\mathcal{W}} := \{\hat{W}_1, \hat{W}_2\}$ with $\hat{W}_i := W_i \cup \{w_i\}$ for $i \in \{1, 2\}$. Like in the „$\Rightarrow$“ part $\hat{\mathcal{W}}$ is a solution for $\hat{\mathcal{I}}$.

Thus, FFPc is $\mathbb{NP}$-complete. $\square$

The last two proofs show that the transformations from FPPc to FPP are not of structural difficulty. The main idea is adding the centers of an instance of FPPc to an instance of FPP and extending the subset function canonically. Because of this in the following section just algorithms for FPP are considered. Instances of FPPc can be handled in the same way.

### 3.1.1 Heuristic approach

Since FPP is $\mathbb{NP}$-complete, a heuristic is used to solve the problem. To create such a heuristic some structure on $p$ is needed. Thus, in this section $p$ is assumed to be monotonically decreasing and $P$ is assumed to be monotonically increasing. The idea is to start with some random partition of $V$ and iteratively modify it until a feasible partition is found. In the following definition an ILP is introduced. This program creates an incidence vector for a new partition, which is used as partition in the next iteration. To avoid loops in this process one special partitioning set (indexed by $j$) is selected. The target function is designed such that moving elements to a different partitioning set is punished except if the destination set is $W_j$. The key result of this section is that loops cannot occur if the program can be solved with optimal solution value 0.

**Definition 3.1.7**
*Let $\mathcal{I} = (V, p, k, b)$ be an instance of FPP, $\mathcal{W} = (W_1, \dots, W_k)$ a partition of $V$ and $j \in \{1, \dots, k\}$. The problem $FPPlin_{\mathcal{I},\mathcal{W},j}$ is defined as the following ILP:*

$$\min \sum_{\substack{i=1 \\ i \neq j}}^{k} \sum_{v \notin W_i} x_{i,v}$$

$$s.t.$$

$$\sum_{i=1}^{k} x_{i,v} = 1 \qquad \forall v \in V$$

$$\sum_{v \in V} p(W_i \cup \{v\}, v) x_{i,v} \leq b_i \qquad \forall i \in \{1, \dots, k\}$$

$$x_{i,v} \in \{0, 1\} \qquad \forall i \in \{1, \dots, k\}, v \in V$$

The incidence vector $x$ encodes the membership of elements of $V$ to a partitioning set. That means $x_{i,v} = 1$ if $v \in V$ belongs to partitioning set $W_i$. The coefficients $p(W_i \cup \{v\}, v)$ are used to approximate the weight of $v$ if $v$ is moved to $W_i$. Generally, this is not the exact weight since other elements could also be added to $W_i$ or removed from $W_i$.

The following definition describes how a family of subsets of $V$ is created from $x$.

**Definition 3.1.8**
*Let $V$ be a finite set, $k \in \mathbb{N}$ and $x \in \{0,1\}^{\{1,\dots,k\}\times V}$. For $i \in \{1,\dots,k\}$ define*

$$W_i^x := \{v \in V : x_{i,v} = 1\}$$

*and*

$$\mathcal{W}^x := \{W_1^x, \dots, W_k^x\}.$$

Obviously, $\mathcal{W}^x$ forms a partition of $V$ if $\sum_{i=1}^k x_{i,v} = 1$ for all $v \in V$.

**Lemma 3.1.9**
*Let $V$ be a finite set, $k \in \mathbb{N}$ and $x \in \{0,1\}^{\{1,\dots,k\}\times V}$. If*

$$\sum_{i=1}^k x_{i,v} = 1$$

*for all $v \in V$, then $\mathcal{W}^x$ is a partition of $V$.*

*Proof*
For each $v \in V$ there exists exactly one $i_v \in \{1,\dots,k\}$ with $x_{i_v,v} = 1$. This means $v \in W_{i_v}^x$ and $v \notin W_j^x$ for all $j \in \{1,\dots,k\}\backslash\{i_v\}$. Thus, $\mathcal{W}^x$ is a partition of $V$. $\square$

If $\text{FPPlin}_{\mathcal{I},\mathcal{W},j}$ can be solved with optimal solution value 0, the following lemma guarantees that $P(W_j^x) \le b_j$ and that the subset values of all other partitioning sets do not increase. Thus, in this case iteratively solving $\text{FPPlin}_{\mathcal{I},\mathcal{W},j}$ and updating $\mathcal{W}$ with $\mathcal{W}^x$ cannot lead to a loop.

**Lemma 3.1.10**
*Let $\mathcal{I} = (V,p,k,b)$ be an instance of FPP, where $p$ is monotonically decreasing and $P$ is monotonically increasing, $\mathcal{W} = \{W_1,\dots,W_k\}$ be a partition of $V$ and $j \in \{1,\dots,k\}$. If $\mathit{FPPlin}_{\mathcal{I},\mathcal{W},j}$ is solvable with solution $x$ and optimal solution value 0, then $P(W_j^x) \le b_j$ and $P(W_i^x) \le P(W_i)$ for all $i \in \{1,\dots,k\}\backslash\{j\}$.*

*Proof*
Let $i \in \{1, \ldots, k\} \backslash \{j\}$ and $v \notin W_i$. Since

$$\sum_{\substack{i=1 \\ i \neq j}}^{k} \sum_{v \notin W_i} x_{i,v} = 0,$$

it is $x_{i,v} = 0$ and therefore $v \notin W_i^x$. This means $W_i^x \subseteq W_i$. Thus, for all $i \in \{1, \ldots, k\} \backslash \{j\}$ it follows

$$P(W_i^x) \leq P(W_i).$$

Since $\mathcal{W}$ and $\mathcal{W}^x$ are partitions of $V$ and $W_i^x \subseteq W_i$ for $i \in \{1, \ldots, k\} \backslash \{j\}$, it follows $W_j \subseteq W_j^x$. Thus, also for all $w \in W_j^x$ it is $W_j \cup \{w\} \subseteq W_j^x$. The claim follows with

$$\begin{aligned} P(W_j^x) &= \sum_{w \in W_j^x} p(W_j^x, w) \\ &\leq \sum_{w \in W_j^x} p(W_j \cup \{w\}, w) \\ &= \sum_{v \in V} p(W_j \cup \{v\}, v) x_{j,v} \\ &\leq b_j. \end{aligned}$$

□

**Remark 3.1.11**
If $\text{FPPlin}_{\mathcal{I},\mathcal{W},j}$ is solvable with an optimal solution value of 0, then with the previous Lemma 3.1.10 it is $P(W_j^x) \leq b_j$. However, $P(W_i^x) \leq b_i$ is not guaranteed for $i \in \{1, \ldots, k\} \backslash \{j\}$ even if the inequalities of the program may suggest this. Consider the following example:

Let $\mathcal{I} = (V, p, k, b)$ be an instance of FPP with $V := \{v_1, v_2\}$ and

$$p(W, v) := \begin{cases} \frac{1}{2} & \text{if } W = V \\ \frac{3}{4} & \text{if } W = \{v_1\} \vee W = \{v_2\} \\ 1 & \text{if } W = \emptyset \end{cases}$$

for $W \subseteq V$ and $v \in V$. Further, let $k := 2$, $b_1 := \frac{3}{4}$ and $b_2 := \frac{1}{2}$. The possible subset values are $P(V) = 1$, $P(\{v_1\}) = P(\{v_2\}) = \frac{3}{4}$ and $P(\emptyset) = 0$. It is easy to see that $p$ is monotonically decreasing and $P$ is monotonically increasing. As starting partition select $\mathcal{W} := \{W_1, W_2\}$ with $W_1 := \emptyset$ and $W_2 := V$ and choose $j := 1$.
The $x$ vector with the components $x_{1,v_1} = 1$, $x_{1,v_2} = 0$, $x_{2,v_1} = 0$, $x_{2,v_2} = 1$ is a solution for $\mathrm{FPPlin}_{\mathcal{I},\mathcal{W},j}$ since

$$\sum_{i=1}^{k} x_{i,v} = 1$$

for $v \in \{v_1, v_2\}$ and

$$\begin{aligned}\sum_{v\in V} p(W_1 \cup \{v\}, v)x_{1,v} &= \sum_{v\in V} p(\{v\}, v)x_{1,v} = \frac{3}{4}\sum_{v\in V} x_{1,v} = \frac{3}{4} = b_1,\\ \sum_{v\in V} p(W_2 \cup \{v\}, v)x_{2,v} &= \sum_{v\in V} p(V, v)x_{2,v} = \frac{1}{2}\sum_{v\in V} x_{2,v} = \frac{1}{2} = b_2.\end{aligned}$$

The value of this solution is 0:

$$\sum_{\substack{i=1\\ i\neq j}}^{k} \sum_{v\notin W_i} x_{i,v} = \sum_{\substack{i=1\\ i\neq 1}}^{2} \sum_{v\notin W_i} x_{i,v} = \sum_{v\notin W_2} x_{i,v} = \sum_{v\notin V} x_{i,v} = 0$$

It is $\mathcal{W}^x = \{\{v_1\}, \{v_2\}\}$ and $P(W_1^x) = P(\{v_1\}) = \frac{3}{4} = b_1$, but $P(W_2^x) = P(\{v_2\}) = \frac{3}{4} > b_2$. Thus, $x$ is a solution for $\mathrm{FPPlin}_{\mathcal{I},\mathcal{W},j}$ with value 0 but $\mathcal{W}^x$ is not feasible for $\mathcal{I}$.

Due to Lemma 3.1.10 solving $\mathrm{FPPlin}_{\mathcal{I},\mathcal{W},j}$ with an optimal solution value of 0 creates a partition with $P(W_j^x) \leq b_j$ and $P(W_i^x) \leq P(W_i)$ for all $i \in \{1, \ldots, k\}\backslash\{j\}$. Thus, after one iteration with solution value 0 the subset $W_j$ already fulfills the feasibility criterion. This means after $k$ iterations with solution value 0 and $j = 1$, $j = 2$ to $j = k$ a feasible solution is found.
However, in the iterative process $\mathrm{FPPlin}_{\mathcal{I},\mathcal{W},j}$ may become infeasible. This leads to the termination of the process without finding a solution. In the following a more general approach is presented since $P(W_j^x) \leq b_j$ is not necessarily required after one iteration. Already $P(W_j^x) < P(W_j)$ would be sufficient to ensure a descent in the subset value of $W_j$ and in this way avoid the occurrence of loops. Like this the chance of generating infeasible instances is reduced.

**Definition 3.1.12**
*Let $\mathcal{I} = (V, p, k, b)$ be an instance of FPP, $\mathcal{W} = (W_1, \ldots, W_k)$ a partition of $V$, $j \in \{1, \ldots, k\}$ and $\varepsilon \in \mathbb{R}^+$. The problem FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ is defined as the following ILP:*

$$\min \sum_{\substack{i=1\\ i\neq j}}^{k} \sum_{v \notin W_i} x_{i,v}$$

$$s.t.$$

$$\begin{aligned} \sum_{i=1}^{k} x_{i,v} &= 1 && \forall v \in V \\ \sum_{v\in V} p(W_i \cup \{v\}, v) x_{i,v} &\leq \max\{b_i, P(W_i) - \varepsilon\} && \forall i \in \{1, \ldots, k\} \\ x_{i,v} &\in \{0,1\} && \forall i \in \{1, \ldots, k\}, v \in V \end{aligned}$$

For FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ an equivalent result as Lemma 3.1.10 follows directly.

**Lemma 3.1.13**
*Let $\mathcal{I} = (V, p, k, b)$ be an instance of FPP, where $p$ is monotonically decreasing and $P$ is monotonically increasing, $\mathcal{W} = (W_1, \ldots, W_k)$ be a partition of $V$, $j \in \{1, \ldots, k\}$ and $\varepsilon \in \mathbb{R}^+$. If FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ is solvable with solution $x$ and optimal solution value $0$, then $P(W_j^x) \leq \max\{b_j, P(W_j) - \varepsilon\}$ and $P(W_i^x) \leq P(W_i)$ for all $i \in \{1, \ldots, k\} \backslash \{j\}$.*

*Proof*
Like the proof of Lemma 3.1.10. Substitue $b_j$ by $\max\{b_j, P(W_j) - \varepsilon\}$. □

The result of Lemma 3.1.13 directly leads to the definition of Algorithm 3.1.14.

The following theorem states that Algorithm 3.1.14 terminates if from some point all programs FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ can be solved with solution value 0.

**Theorem 3.1.15**
*Let $\mathcal{I} = (V, p, k, b)$ be an instance of FPP, where $p$ is monotonically decreasing and $P$ is monotonically increasing, $\mathcal{W} = (W_1, \ldots, W_k)$ a partition of $V$ and $\varepsilon \in \mathbb{R}^+$. If Algorithm 3.1.14 is started with $\mathcal{I}$, $\mathcal{W}$ and $\varepsilon$ as input and from some point all programs FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ are solvable with optimal solution value $0$, then Algorithm 3.1.14 terminates and returns a feasible solution for FPP.*

**Algorithm 3.1.14** FPP find solution

**Input:**
- Instance $\mathcal{I} = (V, p, k, b)$ of FPP
- Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$
- $\varepsilon \in \mathbb{R}^+$

**Output:** Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ with $P(W_i) \leq b_i$ for $i \in \{1, \ldots, k\}$

```
1: while $\exists i \in \{1, \ldots, k\} : P(W_i) > b_i$ do
2:     for $j = 1 \to k$ do
3:         if FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ is infeasible then
4:             error: no solution found
5:         else
6:             let $x^*$ be an optimal solution for FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$
7:             $\mathcal{W} \leftarrow \mathcal{W}^{x^*}$
8:         end if
9:     end for
10: end while
11: return $\mathcal{W}$
```

*Proof*
If Algorithm 3.1.14 terminates, it returns a partition of $V$ (Lemma 3.1.9). Further, $P(W_i) \leq b_i$ for all $i \in \{1, \ldots, k\}$ since otherwise the `while` loop (line 1) would not break. Thus, if Algorithm 3.1.14 terminates, a feasible solution for FPP is returned.
To show that Algorithm 3.1.14 terminates assume that FPPlin$_{\mathcal{I},\mathcal{W},j,\varepsilon}$ in line 6 always is solvable with optimal solution value 0. Thus, with Lemma 3.1.13 in each step of the `for` loop (line 2) it is $P(W_j^{x^*}) \leq b_j$ or $P(W_j^{x^*}) \leq P(W_j) - \varepsilon$ and $P(W_i^{x^*}) \leq P(W_i)$ for $i \in \{1, \ldots, k\} \backslash \{j\}$. Therefore, after a complete cycle through the `for` loop the subset value $P(W_i)$ decreases by $\varepsilon$ or it is $P(W_i) \leq b_i$ for all $i \in \{1, \ldots, k\}$. This means after a finite number cycles through the `while` loop (line 1) it is $P(W_i) \leq b_i$ for all $i \in \{1, \ldots, k\}$ and Algorithm 3.1.14 terminates. □

### 3.1.2 Numerical results

To test Algorithm 3.1.14 feasible instances of FPP with 4 different sizes of $V$ and 4 different values for $k$ are constructed. For each instance the runtime of Algorithm 3.1.14 is measured, where as starting partition some random partition is taken.

As comparison a randomized algorithm is implemented. This algorithm randomly generates partitions and returns a feasible partition if one is found. To ensure the comparability of the results the random algorithm is just run for the time Algorithm 3.1.14 needs to solve the instance. The randomized algorithm is run for 100 times. In this way the expected ratio of finding a solution in the same time as Algorithm 3.1.14 can be determined.
The test instances are generated as follows. For each element $v \in V$ an absolute and a relative weight value

$$a_v \in \left\{ \left\lfloor \frac{|V|}{2} \right\rfloor + 1, \left\lfloor \frac{|V|}{2} \right\rfloor + 2, \dots, |V| \right\}$$

and

$$r_v \in \left\{ 1, 2, \dots, \left\lfloor \frac{|V|}{2} \right\rfloor \right\}$$

are randomly generated. The ranges are chosen dependent on $V$ to get a diverse set of weights. Further, note that $\max_{v \in V} r_v \leq a_v$. The subset function on $V$ is defined as

$$p(W, v) := \begin{cases} a_v + \dfrac{r_v}{|W|} & \text{if } W \neq \emptyset \\ a_v + r_v & \text{otherwise} \end{cases}$$

for $W \subseteq V$ and $v \in V$. Obviously, $p$ is monotonically decreasing and it is

$$P(W) = \sum_{w \in W} a_w + \frac{\sum_{w \in W} r_w}{|W|}$$

for $\emptyset \neq W \subseteq V$. With

$$\frac{\sum_{w \in W} r_w}{|W|} \leq \frac{\sum_{w \in W} \max_{v \in V} r_v}{|W|} = \max_{v \in V} r_v \leq a_v$$

for all $W \subseteq V$ and $v \in V$ it follows

$$P(W') = \sum_{w \in W'} a_w + \frac{\sum_{w \in W'} r_w}{|W'|} \leq \sum_{w \in W} a_w \leq P(W)$$

for all $\emptyset \neq W' \subsetneq W \subseteq V$. Thus, $P$ is monotonically increasing.

The value of the bounding vector $b$ is chosen as

$$\left\lceil \frac{\sum_{v\in V} a_v}{k} + \frac{\sum_{v\in V} r_v}{|V|} \right\rceil$$

and such that all components are equal. Thus,

$$b_1 = b_2 = \ldots = b_k = \left\lceil \frac{\sum_{v\in V} a_v}{k} + \frac{\sum_{v\in V} r_v}{|V|} \right\rceil .$$

The expression

$$\frac{\sum_{v\in V} a_v}{k} + \frac{\sum_{v\in V} r_v}{|V|}$$

represents an estimation of the average subset value if the elements could be distributed fractionally among the partitioning sets. However, since the distribution is not fractional this may produce infeasible instances. If Algorithm 3.1.14 cannot find a feasible partition for an instance within 15 minutes, the instance is assumed to be infeasible and the value of $b$ is increased iteratively by 1 until a feasible partition can be found. This practice does not give Algorithm 3.1.14 any structural advantage over the randomized approach because in both cases the same instance with the same $b$ is solved.
Instances with four different sizes of $V$ (20, 40, 60 and 80) and four different values of $k$ (2, 3, 5 and 10) are generated. The values are chosen such that the instances can be solved within 60 seconds. Of course in practical applications often much larger problems occur. However, already for these small problems trends in runtimes and the performance of the randomized algorithms can be observed.

In Table 3.1 the results for the generated test instances are presented. The first two columns contain the size of $V$ and the number of the partitioning sets. In the third column the runtime of the heuristic (Algorithm 3.1.14) is shown. This is also the time after which the randomized algorithm is terminated. The ratio with which the randomized algorithm can find a feasible solution, i.e. how often in the 100 runs a solution is found in the given time, is illustrated in the last column.
For $|V| = 20$ and $|V| = 40$ a solution can be randomly found in 100% of the runs if $k = 2$. Also for $|V| = 20$ and $k = 3$ a ratio of 95% can be achieved. However,

| \|V\| | k | FPP | |
|---|---|---|---|
| | | heur. time | rand. ratio |
| 20 | 2 | 0.218 | 100% |
| | 3 | 0.062 | 95% |
| | 5 | 1.442 | 0% |
| | 10 | 5.513 | 5% |
| 40 | 2 | 0.453 | 100% |
| | 3 | 0.463 | 8% |
| | 5 | 0.787 | 0% |
| | 10 | 11.368 | 0% |
| 60 | 2 | 0.297 | 32% |
| | 3 | 0.486 | 0% |
| | 5 | 2.404 | 0% |
| | 10 | 37.533 | 0% |
| 80 | 2 | 1.047 | 0% |
| | 3 | 1.656 | 0% |
| | 5 | 4.093 | 0% |
| | 10 | 40.789 | 0% |

Table 3.1: FPP results

while $|V|$ and $k$ increase the ratio decreases drastically. For $k = 5$ and $k = 10$ nearly never a solution can be found at all. For $|V| \geq 60$ random solutions are found only for $k = 3$ with a ratio of 32%. This is an indicator of the hardness of FPP and shows that the complexity increases strongly with increasing $|V|$ or $k$. Nevertheless, Algorithm 3.1.14 solves the instances in acceptable time. Indeed, the runtime also increases with increasing $|V|$ or $k$ but for $k \in \{2, 3, 5\}$ all instances can still be solved in less than 10 seconds. For $|V| = 80$ and $k = 10$ a solution still can be found in about 40 seconds.

## 3.2 Minimal Partition Problems

After analyzing a decision type partition problem with nonconstant weight functions in this section two minimization type problems are studied. In contrast to the previous results for these problems the monotonic properties of $p$ and $P$ make a difference in the hardness. To specify the problems two target functions are defined.

**Definition 3.2.1**
*Let $V$ be a finite set and $p$ be a subset function on $V$. For a set $\mathcal{W}$ of subsets of $V$ the value*

$$s_p(\mathcal{W}) := \sum_{W \in \mathcal{W}} P(W)$$

*is called* sum subset value *and*

$$m_p(\mathcal{W}) := \max_{W \in \mathcal{W}} P(W)$$

*is called* maximum subset value *of $\mathcal{W}$ for subset function $p$.*

If $P$ is monotonically increasing, a lower bound for the maximum subset value can be given. This is useful in some proofs later on.

**Lemma 3.2.2**
*Let $\mathcal{W}$ be a partition of a finite set $V$ and $p$ a subset function on $V$, where $P$ is monotonically increasing. Then*

$$m_p(\mathcal{W}) \geq \max_{v \in V} p(\{v\}, v).$$

*Proof*
Since $P$ is monotonically increasing, it follows $P(W) \geq P(\{w\})$ for all $W \subseteq V$ and $w \in W$. Thus,

$$\begin{aligned} m_p(\mathcal{W}) &= \max_{W\in\mathcal{W}} P(W) \\ &\geq \max_{W\in\mathcal{W}} \max_{w\in W} P(\{w\}) \\ &= \max_{W\in\mathcal{W}} \max_{w\in W} p(\{w\}, w) \\ &= \max_{v\in V} p(\{v\}, v). \end{aligned}$$

□

Now a partition problem can be defined in which the sum of the subset values of all partitioning sets has to be minimized.

**Definition 3.2.3** (MinSum Partition Problem)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$ and $k \in \mathbb{N}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$, where $s_p(\mathcal{W})$ is minimal, is called* MinSum Partition Problem *(MinSumPP).*

The following theorem shows that MinSumPP cannot be approximated in polynomial time if $\mathbb{P} \neq \mathbb{NP}$, because otherwise NPP could be solved in polynomial time.

**Theorem 3.2.4**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP with approximation ratio $f$ for some function $f : \mathcal{X} \to [1, \infty)$, where $\mathcal{X}$ is the set of possible instances of MinSumPP, even if $p$ is $\{0,1\}$-valued and $k = 2$.*

*Proof*
For a given instance $\mathcal{I} = (S)$ of NPP let $\overline{S} := \sum_{s\in S} s$. Define an instance $\mathcal{J} := (V, p, k)$ of MinSumPP with $V := S$ and

$$p(W, v) := \begin{cases} 0 & \text{if } \sum_{w\in W} w = \frac{\overline{S}}{2} \\ 1 & \text{otherwise} \end{cases},$$

where $W \subseteq V$ and $v \in V$. Obviously, $p$ is polynomially computable in the size of $V$. Further let $k := 2$.

Like in the proof of Theorem 3.1.2 if follows that the optimal solution of $\mathcal{J}$ has a sum subset value of 0 if and only if $\mathcal{I}$ is solvable.
Assumed a polynomial-time $f$-approximation $\mathcal{A}$ for MinSumPP existed, then $\mathcal{A}$ would return at most $f(\mathcal{J}) \cdot 0 = 0$ for $\mathcal{J}$ if $\mathcal{I}$ was solvable and a value strictly greater than 0 if $\mathcal{I}$ was not solvable. Thus, $\mathcal{I}$ can be solved in polynomial time by solving $\mathcal{J}$ with $\mathcal{A}$. This is in contradiction to the hardness of NPP. □

In the previous theorem the premises appear to be artificially designed to get the bad approximation result. The optimal solution of the constructed instance of MinSumPP has value 0 if the corresponding instance of NPP is solvable. Therefore any approximation algorithm must return 0 if the corresponding instance of NPP is solvable. Thus, restricting $p$ to be positive would be interesting. However, before this case is studied note the following lemma.

**Lemma 3.2.5**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinSumPP, where $p$ is nonnegative and monotonically decreasing. Then $\mathcal{W}^* := \{W_1^*, \ldots, W_k^*\}$ with $W_1^* := V$ and $W_i^* = \emptyset$ for $i \in \{2, \ldots, k\}$ is an optimal solution for $\mathcal{I}$.*

*Proof*
Let $\mathcal{W} = \{W_1, \ldots, W_k\}$ be some partition of $V$. With Lemma 3.0.4 it follows

$$P(W_1) + P(W_2) \geq P(W_1 \cup W_2) = P(W_1 \cup W_2) + 0 = P(W_1 \cup W_2) + P(\emptyset).$$

Thus, moving all elements from $W_2$ to $W_1$ leaves $s_p(\mathcal{W})$ unchanged or even reduces the value. The claim follows by repeating this procedure for all $W_i$ with $i \in \{3, \ldots, k\}$. □

This means that if $p$ is monotonically decreasing, the partition, where all elements of $V$ are contained in one partitioning set and all other sets are empty, is an optimal solution for MinSumPP. To exclude this trivial case the following problem is defined.

**Definition 3.2.6** (Nonempty MinSum Partition Problem)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$ and $k \in \mathbb{N}$, with $|V| \geq k$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ with $W_i \neq \emptyset$ for $i \in \{1, \ldots, k\}$, where $s_p(\mathcal{W})$ is minimal, is called* Nonempty MinSum Partition Problem *(MinSumPP$_{\neq\emptyset}$).*

For MinSumPP$_{\neq\emptyset}$ an equivalent result like for MinSumPP can be shown. The proof is very similar to the proof of Theorem 3.2.4 except the constructed subset function has to be monotonically decreasing and the resulting partitioning sets are not allowed to be empty.

**Theorem 3.2.7**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP$_{\neq\emptyset}$ with approximation ratio $f$ for some function $f : \mathcal{X} \to [1, \infty)$, where $\mathcal{X}$ is the set of possible instances of MinSumPP$_{\neq\emptyset}$, even if $p$ is monotonically decreasing, $\{0, 1\}$-valued and $k = 2$.*

*Proof*
For a given instance $\mathcal{I} = (S)$ of NPP let $\overline{S} := \sum_{s\in S} s$. Without loss of generality let $|S| \geq 2$. Define an instance $\mathcal{J} := (V, p, k)$ of MinSumPP$_{\neq\emptyset}$ with $V := S$ and

$$p(W, v) := \begin{cases} 0 & \text{if } \sum_{w\in W} w \geq \frac{\overline{S}}{2} \\ 1 & \text{otherwise} \end{cases},$$

where $W \subseteq V$ and $v \in V$. Obviously, $p$ is polynomially computable in the size of $V$. Further let $k := 2 \leq |V|$. The subset function $p$ is monotonically decreasing:

Let $W' \subseteq W \subseteq V$ and $v \in V$:

- Case $\sum_{w\in W'} w < \frac{\overline{S}}{2}$:

$$p(W', v) = 1 \geq p(W, v) \in \{0, 1\}$$

- Case $\sum_{w\in W'} w \geq \frac{\overline{S}}{2}$:
  It is $\sum_{w\in W} w \geq \sum_{w\in W'} w \geq \frac{\overline{S}}{2}$ and therefore

$$p(W', v) = 0 = p(W, v)$$

The optimal solution for $\mathcal{J}$ has a sum subset value of 0 if and only if $\mathcal{I}$ is solvable:

„$\Rightarrow$“: Let $\mathcal{W} = \{W_1, W_2\}$ be a feasible solution for $\mathcal{J}$ with $W_i \neq \emptyset$ for $i \in \{1, 2\}$ and $s_p(\mathcal{W}) = 0$. Since

$$0 \leq P(W_1) \leq s_p(\mathcal{W}) = 0$$

and

$$0 \leq P(W_2) \leq s_p(\mathcal{W}) = 0,$$

it follows $p(W_i, v) = 0$ for all $v \in W_i$ and $i \in \{1, 2\}$. Since $W_1 \neq \emptyset$, it follows

$$\begin{aligned}
&\sum_{w \in W_1} w \geq \frac{\overline{S}}{2} \\
\Leftrightarrow\ & 2 \sum_{w \in W_1} w \geq \overline{S} \\
\Leftrightarrow\ & 2 \sum_{w \in W_1} w \geq \sum_{s \in S} s = \sum_{v \in V} v \\
\Leftrightarrow\ & \sum_{w \in W_1} w \geq \sum_{v \in V} v - \sum_{w \in W_1} w = \sum_{v \in V \setminus W_1} v = \sum_{v \in W_2} w.
\end{aligned}$$

Because $W_2 \neq \emptyset$ in the same way

$$\sum_{w \in W_2} w \geq \sum_{w \in W_1} w$$

follows and therefore

$$\sum_{w \in W_1} w = \sum_{w \in W_2} w.$$

Then $\mathcal{T} := \{T_1, T_2\}$ with $T_1 := W_1$ and $T_2 := W_2$ is a valid partition of $S$ for $\mathcal{I}$.

„$\Leftarrow$“: Let $\mathcal{T} = \{T_1, T_2\}$ be a partition of $S$ with

$$\sum_{s \in T_1} s = \sum_{s \in T_2} s.$$

Since $|S| \geq 2$, it is $T_i \neq \emptyset$ for $i \in \{1, 2\}$. With Lemma 1.2.3 it follows

$$\sum_{s \in T_1} s = \sum_{s \in T_2} s = \frac{\overline{S}}{2}.$$

Define $W_i := T_i \neq \emptyset$ for $i \in \{1, 2\}$. Then

$$P(W_i) = \sum_{v \in W_i} p(W_i, v) = \sum_{v \in W_i} 0 = 0$$

for $i \in \{1, 2\}$ and $\{W_1, W_2\}$ is a valid partition of $V$ for $\mathcal{J}$ with a sum subset value of 0.

The claim follows like in the proof of Theorem 3.2.4. □

As already mentioned before, Theorems 3.2.4 and 3.2.7 seem to be designed in a way to get bad approximation results since the optimal value of MinSumPP or MinSumPP$_{\neq\emptyset}$ is 0. Thus, the following two theorems cover the case with a positive subset function $p$ for which an optimal value of 0 is impossible.
In Theorem 3.2.8 for MinSumPP$_{\neq\emptyset}$ the subset function $p$ is assumed to be monotonically decreasing and in Theorem 3.2.9 for MinSumPP the cumulated subset function $P$ is assumed to be monotonically increasing.

**Theorem 3.2.8**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP$_{\neq\emptyset}$ with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$ and $p$ is positive, monotonically decreasing and only takes two values.*

*Proof*
Let $\mathcal{I} = (S)$ be an instance of NPP and $\overline{S} := \sum_{s \in S} s$. Without loss of generality let $|S| \geq 2$. Assumed a polynomial-time $M$-approximation algorithm $\mathcal{A}$ for MinSumPP$_{\neq\emptyset}$ existed with $M \geq 1$. Define an instance $\mathcal{J} := (V, p, k)$ of MinSumPP$_{\neq\emptyset}$ with $V := S$,

$$p(W, v) := \begin{cases} \varepsilon & \text{if } \sum_{w \in W} w \geq \frac{\overline{S}}{2} \\ M|V| & \text{otherwise} \end{cases},$$

for $W \subseteq V$, $v \in V$, $\varepsilon \in (0, \frac{1}{2})$ and $k := 2 \leq |V|$. Obviously, $p$ is polynomially computable in the size of $V$ and positive.

Further, $p$ is monotonically decreasing:

Let $W' \subseteq W \subseteq V$ and $v \in V$:

- Case $\sum_{w \in W'} w < \frac{\overline{S}}{2}$:
$$p(W', v) = M|V| \geq p(W, v) \in \{\varepsilon, M|V|\}$$
- Case $\sum_{w \in W'} w \geq \frac{\overline{S}}{2}$:
It is $\sum_{w \in W} w \geq \sum_{w \in W'} w \geq \frac{\overline{S}}{2}$ and therefore
$$p(W', v) = \varepsilon = p(W, v)$$

It is

$$P(W) = \sum_{v \in W} p(W, v) = \begin{cases} \varepsilon|W| & \text{if } \sum_{w \in W} w \geq \frac{\overline{S}}{2} \\ M|V||W| & \text{otherwise} \end{cases}.$$

For a solution $\mathcal{W} = \{W_1, W_2\}$ of $\mathcal{J}$ with

$$\sum_{w \in W_1} w = \sum_{w \in W_2} w = \frac{\overline{S}}{2}$$

it follows $P(W_i) = \varepsilon|W_i| \leq \varepsilon|V|$ for $i \in \{1, 2\}$ and therefore $s_p(\mathcal{W}) \leq 2\varepsilon|V| < |V|$. For a solution $\mathcal{W} = \{W_1, W_2\}$ of $\mathcal{J}$ with $\sum_{w \in W_1} w \neq \sum_{w \in W_2} w$ (without loss of generality $\sum_{w \in W_1} w > \sum_{w \in W_2} w$) it is

$$\begin{aligned} & 2\sum_{w \in W_2} w = \sum_{w \in W_2} w + \overline{S} - \sum_{w \in W_1} w < \sum_{w \in W_1} w + \overline{S} - \sum_{w \in W_1} w = \overline{S} \\ \Leftrightarrow & \sum_{w \in W_2} w < \frac{\overline{S}}{2}. \end{aligned}$$

Therefore, $s_p(\mathcal{W}) \geq P(W_2) = M|V||W_2| \geq M|V|$ because $W_2 \neq \emptyset$.
Altogether, since $\mathcal{A}$ is a polynomial-time $M$-approximation, a value less than $M|V|$ is returned by $\mathcal{A}$ for $\mathcal{J}$ if and only if $\mathcal{I}$ is solvable and a value larger or equal $M|V|$ if and only if $\mathcal{I}$ is infeasible. Setting $M := 2^{\text{poly}(|V|, k)}$ with some polynomial poly : $\mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ leaves the transformation polynomial in the coding length of the input. This proves the claim. $\square$

**Theorem 3.2.9**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$, $p$ is positive and only takes two values and $P$ is monotonically increasing.*

*Proof*
Let $\mathcal{I} = (S)$ be an instance of NPP and $\overline{S} := \sum_{s \in S} s$. Without loss of generality let $|S| \geq 2$. Assumed a polynomial-time $M$-approximation algorithm $\mathcal{A}$ for MinSumPP existed with $M \geq 1$. Define an instance $\mathcal{J} := (V, p, k)$ of MinSumPP with $V := S$,

$$p(W, v) := \begin{cases} \varepsilon & \text{if } \sum_{w \in W} w \leq \frac{\overline{S}}{2} \\ M|V| & \text{otherwise} \end{cases},$$

for $W \subseteq V$, $v \in V$, $\varepsilon \in (0, \frac{1}{2})$ and $k := 2$. Obviously, $p$ is polynomially computable in the size of $V$ and positive.
Further, $P$ is monotonically increasing:

- Case $\sum_{w \in W'} w \leq \frac{\overline{S}}{2}$:
$$p(W', v) = \varepsilon \leq p(W, v) \in \{\varepsilon, M|V|\}$$
- Case $\sum_{w \in W'} w > \frac{\overline{S}}{2}$:
It is $\sum_{w \in W} w \geq \sum_{w \in W'} w > \frac{\overline{S}}{2}$ and therefore
$$p(W', v) = M|V| = p(W, v)$$

Similar as in the proof of Theorem 3.2.8 it is

$$P(W) = \sum_{v \in W} p(W, v) = \begin{cases} \varepsilon|W| & \text{if } \sum_{w \in W} w \leq \frac{\overline{S}}{2} \\ M|V||W| & \text{otherwise} \end{cases}.$$

For a solution $\mathcal{W} = \{W_1, W_2\}$ of $\mathcal{J}$ with

$$\sum_{w \in W_1} w = \sum_{w \in W_2} w = \frac{\overline{S}}{2}$$

it follows $P(W_i) = \varepsilon|W_i| \leq \varepsilon|V|$ for $i \in \{1, 2\}$ and therefore $s_p(\mathcal{W}) \leq 2\varepsilon|V| < |V|$. For a solution $\mathcal{W} = \{W_1, W_2\}$ of $\mathcal{J}$ with $\sum_{w\in W_1} w \neq \sum_{w\in W_2} w$ (without loss of generality $\sum_{w\in W_1} w > \sum_{w\in W_2} w$) it is

$$2\sum_{w\in W_1} w = \sum_{w\in W_1} w + \overline{S} - \sum_{w\in W_2} w > \sum_{w\in W_2} w + \overline{S} - \sum_{w\in W_2} w = \overline{S}.$$

This also means $W_1 \neq \emptyset$. Therefore, $s_p(\mathcal{W}) \geq P(W_1) = M|V||W_1| \geq M|V|$. Altogether, since $\mathcal{A}$ is a polynomial-time $M$-approximation a value less than $M|V|$ is returned by $\mathcal{A}$ for $\mathcal{J}$ if and only if $\mathcal{I}$ is solvable and a value larger or equal $M|V|$ if and only if $\mathcal{I}$ is infeasible. Setting $M := 2^{\text{poly}(|V|,k)}$ with some polynomial poly : $\mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ leaves the transformation polynomial in the coding length of the input. This proves the claim. $\square$

Note that in Theorem 3.2.9 MinSumPP is studied and thus empty partitioning sets are allowed. This is because Lemma 3.2.5 just holds for monotonically decreasing subset functions. Nevertheless, a similar result with the same proof can be obtained for MinSumPP$_{\neq\emptyset}$.
From a practical point of view Theorems 3.2.8 and 3.2.9 are not very promising. Even if either $p$ is monotonically decreasing or if $P$ is monotonically increasing there is no approximation algorithm for MinSumPP$_{\neq\emptyset}$ with an approximation ratio polynomial in the size of the input. On the positive side, for the case where both monotonic properties hold such an approximation algorithm is presented later on.

But first, like in the previous section, MinSumPP and MinSumPP$_{\neq\emptyset}$ can be extended by a set of fixed centers which have to be contained in different partitioning sets.

**Definition 3.2.10** (MinSum Partition Problem with fixed centers)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$ and $w_i \in V$ for $i \in \{1, \dots, k\}$ with $w_i \neq w_j$ for all $i, j \in \{1, \dots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $w_i \in W_i$ for $i \in \{1, \dots, k\}$, where $s_p(\mathcal{W})$ is minimal, is called* MinSum Partition Problem with fixed centers *(MinSumPPc).*

**Definition 3.2.11** (Nonempty MinSum Partition Problem with fixed centers)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$, with $|V| \geq 2k$, and $w_i \in V$ for $i \in \{1, \dots, k\}$ with $w_i \neq w_j$ for all $i, j \in \{1, \dots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $w_i \in W_i$ and $W_i \backslash \{w_i\} \neq \emptyset$ for all $i \in \{1, \dots, k\}$, where $s_p(\mathcal{W})$ is minimal, is called* Nonempty MinSum Partition Problem with fixed centers *(MinSumPPc$_{\neq\emptyset}$).*

For these two problems equivalent results as Theorems 3.2.8 and 3.2.9 can be obtained by a transformation to the corresponding problems without fixed centers.

**Theorem 3.2.12**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPPc$_{\neq\emptyset}$ with approximation ratio $\mathcal{O}\left(2^{\mathrm{poly}(|V|,k)}\right)$ for some polynomial* $\mathrm{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$ and $p$ is positive, monotonically decreasing and only takes three values.*
*Further unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinSumPPc with approximation ratio $\mathcal{O}\left(2^{\mathrm{poly}(|V|,k)}\right)$ for some polynomial* $\mathrm{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$, $p$ is positive, only takes three values and $P$ is monotonically increasing.*

*Proof*
Let $\mathcal{I} = (V, p, k)$ be an instance of MinSumPP$_{\neq\emptyset}$ (or MinSumPP respectively) with positive subset function $p$, $k = 2$ and without loss of generality $|V| > 0$. Define an instance $\hat{\mathcal{I}} = (\hat{V}, \hat{p}, \hat{k}, \hat{w})$ of MinSumPPc$_{\neq\emptyset}$ (or MinSumPPc respectively) with $\hat{V} := V \cup \{\hat{w}_1, \hat{w}_2\}$, where $\hat{w}_1, \hat{w}_2 \notin V$ are two arbitrary elements with $\hat{w}_1 \neq \hat{w}_2$, and

$$\hat{p}(\hat{W}, \hat{v}) := \begin{cases} p(\hat{W} \cap V, \hat{v}) & \text{if } \hat{v} \in V \\ \varepsilon & \text{otherwise} \end{cases},$$

where $\hat{W} \subseteq \hat{V}$, $\hat{v} \in \hat{V}$ and

$$\varepsilon := \min \left\{ \min_{v \in V} p(\{v\}, v), \min_{v \in V} p(V, v) \right\}.$$

Obviously, $\hat{p}$ is polynomially computable in the size of $\hat{V}$ because $p$ is polynomially computable in the size of $V$. It is $\varepsilon > 0$ since $p$ is positive. Thus, $\hat{p}$ is positive.

Further, let $\hat{k} := k = 2$ (if $k \leq |V|$ then $2\hat{k} = 2k = k + k \leq |V| + 2 = |\hat{V}|$). As in the proof of Theorem 3.1.6 $\hat{p}$ is monotonically decreasing if $p$ is positive and monotonically decreasing, and $\hat{P}$ is monotonically increasing if $P$ is monotonically increasing. Also, $\hat{p}$ only takes three values if $p$ only takes two values.
For a feasible partition $\hat{\mathcal{W}} = \{\hat{W}_1, \hat{W}_2\}$ for $\hat{\mathcal{I}}$, define the corresponding partition $\mathcal{W} := \{W_1, W_2\}$ of $V$ for $\mathcal{I}$ with $W_i := \hat{W}_i \backslash \{\hat{w}_i\}$ for $i \in \{1, 2\}$. Thus, $W_i \neq \emptyset$ if $\hat{W}_i \backslash \{\hat{w}_i\} \neq \emptyset$ for $i \in \{1, 2\}$. Like in the proof of Theorem 3.1.6 it is

$$\hat{P}(\hat{W}_i) = P(W_i) + \varepsilon$$

and therefore

$$\begin{aligned} s_p(\mathcal{W}) &= \sum_{i \in \{1,2\}} P(W_i) \\ &< \sum_{i \in \{1,2\}} P(W_i) + k\varepsilon \\ &= \sum_{i \in \{1,2\}} (P(W_i) + \varepsilon) \\ &= s_{\hat{p}}(\hat{\mathcal{W}}). \end{aligned} \tag{3.2.1}$$

If $W_i = \hat{W}_i \backslash \{\hat{w}_i\} \neq \emptyset$, then for positive and monotonically decreasing $p$ it follows

$$\varepsilon \leq \min_{v \in V} p(V, v) \leq \sum_{v \in W_i} p(V, v) \leq \sum_{v \in W_i} p(W_i, v) = P(W_i)$$

and for positive $p$ and monotonically increasing $P$ it follows

$$\varepsilon \leq \min_{v \in V} p(\{v\}, v) = \min_{v \in V} P(\{v\}) \leq \min_{w \in W_i} P(\{w\}) \leq P(W_i)$$

for $i \in \{1, 2\}$. Thus, in both cases

$$\hat{P}(\hat{W}_i) = P(W_i) + \varepsilon \leq 2P(W_i)$$

if $W_i \neq \emptyset$ for $i \in \{1, 2\}$, which also means

$$s_{\hat{p}}(\hat{\mathcal{W}}) = \sum_{i \in \{1,2\}} \hat{P}(\hat{W}_i) \leq \sum_{i \in \{1,2\}} 2P(W_i) = 2s_p(\mathcal{W}). \tag{3.2.2}$$

If for example $W_1 = \emptyset$, then it is $\hat{W}_1 = \{\hat{w}_1\}$, $W_2 = V \neq \emptyset$ and

$$\begin{aligned} s_{\hat{p}}(\hat{\mathcal{W}}) &= \sum_{i \in \{1,2\}} \hat{P}(\hat{W}_i) \\ &= \hat{P}(\hat{W}_1) + \hat{P}(\hat{W}_2) \\ &= \varepsilon + P(W_2) + \varepsilon \\ &\leq 3P(W_2) \\ &= 3s_p(\mathcal{W}). \end{aligned} \tag{3.2.3}$$

Thus, altogether in any case

$$s_{\hat{p}}(\hat{\mathcal{W}}) \leq 3s_p(\mathcal{W}).$$

Let $z^*$ be the optimal solution value for $\mathcal{I}$ and $\hat{z}$ the value of the corresponding solution for $\hat{\mathcal{I}}$ (centers are removed from all partitioning sets). Note that $\hat{z}$ is not necessarily the optimal value for $\hat{\mathcal{I}}$. It follows

$$3z^* \geq \hat{z}.$$

Let $\mathcal{A}'$ be a polynomial-time $c$-approximation algorithm for some $c \geq 1$ for MinSumPPc$_{\neq\emptyset}$ (or MinSumPPc respectively), $\hat{z}^*$ the optimal solution value for $\hat{\mathcal{I}}$, $\hat{\mathcal{W}}'$ the solution returned by $\mathcal{A}'$ and $\mathcal{W}'$ the corresponding solution for $\mathcal{I}$. Further, let $\hat{z}' := s_{\hat{p}}(\hat{\mathcal{W}}')$ and $z' := s_p(\mathcal{W}')$. Then

$$3z^* \geq \hat{z} \geq \hat{z}^* \geq \frac{1}{c}\hat{z}' > \frac{1}{c}z',$$

where the last inequality follows from Equation (3.2.1). This means $\mathcal{A}'$ is a $3c$-approximation algorithm for MinSumPP$_{\neq\emptyset}$ (or MinSumPP respectively). The claim follows with Theorem 3.2.8 for monotonically decreasing $p$ and with Theorem 3.2.9 for monotonically increasing $P$. $\square$

The next problem studied is the MinMax Partition Problem, which is very similar to the MinSum Partition Problem. In this context not the sum of all subset values has to be minimized but the maximum of all these values. Thus, the problem is defined as follows.

**Definition 3.2.13** (MinMax Partition Problem)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$ and $k \in \mathbb{N}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$, where $m_p(\mathcal{W})$ is minimal, is called* MinMax Partition Problem *(MinMaxPP).*

The complexity results are almost identical to the results for MinSumPP. This is shown in the following theorems.

**Theorem 3.2.14**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPP with approximation ratio $f$ for some function $f : \mathcal{X} \to [1, \infty)$, where $\mathcal{X}$ is the set of possible instances of MinMaxPP, even if $p$ is $\{0,1\}$-valued and $k = 2$.*

*Proof*
For a finite set $V$, a nonnegative subset function $p$ and a set $\mathcal{W}$ of subsets of $V$

$$m_p(\mathcal{W}) = 0 \Leftrightarrow s_p(\mathcal{W}) = 0$$

holds. Thus, the same transformation as in the proof of Theorem 3.2.4 can be used and the claim follows. □

For MinSumPP with monotonically decreasing $p$ it was shown that the trival partition is an optimal solution. A similar result holds for MinMaxPP. If $p$ is monotonically decreasing, already the trivial partition is an approximate solution. However, the approximation ratio depends on the values of $p$.
Later, for monotonically increasing $P$ also an approximation algorithm is presented whose approximation ratio only depends on the size of $V$ and $k$.

**Lemma 3.2.15**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, where $p$ is positive and monotonically decreasing. Further let $\mathcal{W}^* := \{W_1^*, \dots, W_k^*\}$ be an optimal solution for $\mathcal{I}$, $\varepsilon := \max_{v \in V} p(\{v\}, v) > 0$ and $\delta := \min_{v \in V} p(V, v) > 0$. Then*

$$m_p(\mathcal{W}) \leq \frac{\varepsilon}{\delta} |V| m_p(\mathcal{W}^*)$$

*for the partition $\mathcal{W} := \{W_1, \dots, W_k\}$ of $V$ with $W_1 := V$ and $W_i = \emptyset$ for $i \in \{2, \dots, k\}$.*

*Proof*
Without loss of generality assume $V \neq \emptyset$ and $P(W_1^*) \geq \ldots \geq P(W_k^*)$. Then $W_1 \neq \emptyset$ and $m_p(\mathcal{W}^*) = P(W_1^*)$. Further,

$$\begin{aligned} P(W_1^*) &= \sum_{w \in W_1^*} p(W_1^*, w) \\ &\geq \min_{w \in W_1^*} p(W_1^*, w) \\ &\geq \min_{v \in V} p(W_1^*, v) \\ &\geq \min_{v \in V} p(V, v) \\ &= \delta \end{aligned}$$

and

$$P(V) = \sum_{v \in V} p(V, v) \leq \sum_{v \in V} p(\{v\}, v) \leq \sum_{v \in V} \varepsilon = \varepsilon |V|.$$

Thus,

$$m_p(\mathcal{W}) = P(V) \leq \varepsilon |V| = \frac{\varepsilon}{\delta} |V| \delta \leq \frac{\varepsilon}{\delta} |V| P(W_1^*) = \frac{\varepsilon}{\delta} |V| m_p(\mathcal{W}^*).$$

□

To exclude this trivial solution the following definition is given.

**Definition 3.2.16** (Nonempty MinMax Partition Problem)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$ and $k \in \mathbb{N}$, with $|V| \geq k$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ with $W_i \neq \emptyset$ for $i \in \{1, \ldots, k\}$, where $m_p(\mathcal{W})$ is minimal, is called* Nonempty MinMax Partition Problem *(MinMaxPP$_{\neq\emptyset}$).*

All following theorems up to Theorem 3.2.22 are in one-to-one correspondence with the theorems for MinSumPP.

**Theorem 3.2.17**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPP$_{\neq\emptyset}$ with approximation ratio $f$ for some function $f : \mathcal{X} \to [1, \infty)$, where $\mathcal{X}$ is the set of possible instances of MinMaxPP$_{\neq\emptyset}$, even if $p$ is monotonically decreasing, $\{0, 1\}$-valued and $k = 2$.*

*Proof*
For a finite set $V$, a nonnegative subset function $p$ and a set $\mathcal{W}$ of subsets of $V$

$$m_p(\mathcal{W}) = 0 \Leftrightarrow s_p(\mathcal{W}) = 0$$

holds. Thus, the same transformation as in the proof of Theorem 3.2.7 can be used and the claim follows. □

**Theorem 3.2.18**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPP$_{\neq\emptyset}$ with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* poly : $\mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$ and $p$ is positive, monotonically decreasing and only takes two values.*

*Proof*
The proof is analogous to the proof of Theorem 3.2.8 except $\varepsilon \in (0, 1)$ and $s_p(\mathcal{W})$ is substituted by $m_p(\mathcal{W})$. □

**Theorem 3.2.19**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* poly : $\mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$, $p$ is positive and $P$ is monotonically increasing.*

*Proof*
The proof is analogous to the proof of Theorem 3.2.9 except $\varepsilon \in (0, 1)$ and $s_p(\mathcal{W})$ is substituted by $m_p(\mathcal{W})$. □

Like for MinSumPP a similar result of Theorem 3.2.19 holds for MinMaxPP$_{\neq\emptyset}$. In the following two definitions fixed centers are introduced for MinMaxPP.

**Definition 3.2.20** (MinMax Partition Problem with fixed centers)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$ and $w_i \in V$ for $i \in \{1, \dots, k\}$ with $w_i \neq w_j$ for all $i, j \in \{1, \dots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $w_i \in W_i$ for $i \in \{1, \dots, k\}$, where $m_p(\mathcal{W})$ is minimal, is called* MinMax Partition Problem with fixed centers *(MinMaxPPc).*

**Definition 3.2.21** (Nonempty MinMax Partition Problem with fixed centers)
*Let $V$ be a finite set, $p$ a nonnegative subset function on $V$, $k \in \mathbb{N}$, with $|V| \geq 2k$ and $w_i \in V$ for $i \in \{1, \dots, k\}$ with $w_i \neq w_j$ for all $i, j \in \{1, \dots, k\}$.*
*The problem to find a partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ with $w_i \in W_i$ and $W_i \backslash \{w_i\} \neq \emptyset$ for $i \in \{1, \dots, k\}$, such that $m_p(\mathcal{W})$ is minimal, is called* Nonempty MinMax Partition Problem with fixed centers *(MinMaxPPc$_{\neq\emptyset}$).*

Again, also these problems do not admit a polynomial-time approximation algorithm with exponential approximation ratio.

**Theorem 3.2.22**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPPc$_{\neq\emptyset}$ with approximation ratio $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$ and $p$ is positive, monotonically decreasing and only takes three values.*
*Further unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MinMaxPPc with approximation ratio $\mathcal{O}\left(2^{\text{poly}(|V|,k)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}_{\geq 1}$ *even if $k = 2$, $p$ is positive, only takes three values and $P$ is monotonically increasing.*

*Proof*
The proof is analogous to the proof of Theorem 3.2.12 with a few substitutions. Equation (3.2.1) becomes

$$m_p(\mathcal{W}) = \max_{i \in \{1,2\}} P(W_i) < \max_{i \in \{1,2\}} P(W_i) + \varepsilon = \max_{i \in \{1,2\}} (P(W_i) + \varepsilon) = m_{\hat{p}}(\hat{\mathcal{W}}),$$

Equation (3.2.2) becomes

$$m_{\hat{p}}(\hat{\mathcal{W}}) = \max_{i \in \{1,2\}} \hat{P}(\hat{W}_i) \leq \max_{i \in \{1,2\}} 2P(W_i) = 2m_p(\mathcal{W})$$

and Equation (3.2.3) becomes

$$\begin{aligned} m_{\hat{p}}(\hat{\mathcal{W}}) &= \max_{i\in\{1,2\}} \hat{P}(\hat{W}_i) \\ &= \max\{\hat{P}(\hat{W}_1), \hat{P}(\hat{W}_2)\} \\ &= \max\{\varepsilon, P(W_2)+\varepsilon\} \\ &\le 2P(W_2) \\ &= 2s_p(\mathcal{W}). \end{aligned}$$

Altogether then

$$m_{\hat{p}}(\hat{\mathcal{W}}) \le 2m_p(\mathcal{W}).$$

Algorithm $\mathcal{A}'$ is a polynomial-time $c$-approximation algorithm for MinMaxPPc$_{\neq\emptyset}$ (or MinMaxPPc respectively) and thus also is a $2c$-approximation algorithm for MinMaxPP$_{\neq\emptyset}$ (or MinMaxPP respectively). Further, replace $\hat{z}' := m_{\hat{p}}(\hat{\mathcal{W}}')$ and $z' := m_p(\mathcal{W}')$. The claim follows with Theorems 3.2.18 and 3.2.19. $\square$

The last proofs show that the problems MinSumPPc, MinSumPPc$_{\neq\emptyset}$, MinMaxPPc and MinMaxPPc$_{\neq\emptyset}$ are not structurally different from MinSumPP, MinSumPP$_{\neq\emptyset}$, MinMaxPP and MinMaxPP$_{\neq\emptyset}$. Thus, no special algorithms are developed in the next section. All problems can be solved with the methods for the corresponding problem without fixed centers.

### 3.2.1 Heuristic approach

Since in general MinSumPP and MinMaxPP do not admit polynomial-time approximation algorithms, in this section heuristics for the general case are presented. The idea is to start with some partition of $V$ and move single elements from one partitioning set to another or switch two elements such that the target function decreases. For MinSumPP characterizing the descent of the target function is rather easy, whereas for MinMaxPP several cases have to be distinguished.

First, depending on a partition of $V$, two new partitions must be defined. These are generated by moving one element from one partitioning set to another or switching two elements. This is stated in the following two definitions.

**Definition 3.2.23**
*Let $V$ be a finite set and $\mathcal{W} = \{W_1, \ldots, W_k\}$ a partition of $V$ with $k \in \mathbb{N}$. Let $v \in V$ and $j \in \{1, \ldots, k\}$, where $v \notin W_j$. Define*

$$\mathcal{W}_{(v,j)} := \{W_{(v,j)_1}, \ldots, W_{(v,j)_k}\}$$

*with $W_{(v,j)_i} := W_i \backslash \{v\}$, $W_{(v,j)_j} := W_j \cup \{v\}$ and $W_{(v,j)_l} := W_l$ for all $l \in \{1, \ldots, k\} \backslash \{i, j\}$, where $i \in \{1, \ldots, k\}$ such that $v \in W_i$.*
*Further, define the* set of possible moves *for $\mathcal{W}$ as*

$$M(\mathcal{W}) := \{(v, j) \in V \times \{1, \ldots, k\} : v \notin W_j\}.$$

**Definition 3.2.24**
*Let $V$ be a finite set and $\mathcal{W} = \{W_1, \ldots, W_k\}$ a partition of $V$ with $k \in \mathbb{N}$. Let $v, w \in V$, where $|\{v, w\} \cap W| \leq 1$ for all $W \in \mathcal{W}$. Define*

$$\mathcal{W}_{(v,w)} := \{W_{(v,w)_1}, \ldots, W_{(v,w)_k}\}$$

*with $W_{(v,w)_i} := (W_i \backslash \{v\}) \cup \{w\}$, $W_{(v,w)_j} := (W_j \backslash \{w\}) \cup \{v\}$ and $W_{(v,w)_l} := W_l$ for $l \in \{1, \ldots, k\} \backslash \{i, j\}$, where $i, j \in \{1, \ldots, k\}$ such that $v \in W_i$ and $w \in W_j$.*
*Further define the* set of possible switches *for $\mathcal{W}$ as*

$$S(\mathcal{W}) := \{(v, w) \in V^2 : |\{v, w\} \cap W| \leq 1 \ \forall W \in \mathcal{W}\}.$$

**Remark 3.2.25**
Obviously, the two sets $\mathcal{W}_{(v,j)}$ and $\mathcal{W}_{(v,w)}$ are partitions of $V$. The condition $v \notin W_j$ in Definition 3.2.23 implies that $v$ is not already in the partitioning set to which it is moved. Further, the condition $|\{v, w\} \cap W| \leq 1$ in Definition 3.2.24 encodes that $v$ and $w$ must be in different partitioning sets. Thus, $\mathcal{W}_{(v,j)}$ as well as $\mathcal{W}_{(v,w)}$ are different from the original partition $\mathcal{W}$ and differ in exactly two partitioning sets. The sets of possible moves and switches are defined to get a cleaner notation in algorithms.

**MinSumPP**

In the following a function $g_s$ is defined to compare partitions which differ in two partitioning sets. It indicates which of the two partitions has the smaller sum subset value. For MinSumPP this seems to be overloaded in some way but for MinMaxPP such a criterion is useful. Thus, for the sake of conformity the following definition is given.

**Definition 3.2.26**
*For an instance $\mathcal{I} = (V, p, k)$ of MinSumPP and two feasible solutions $\mathcal{W} = \{W_1, \dots, W_k\}$ and $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ for $\mathcal{I}$ with $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ define*

$$g_s(\mathcal{W}, \mathcal{W}') := P(W_i) - P(W'_i) + P(W_j) - P(W'_j),$$

*where $i, j \in \{1, \dots, k\}$ such that $W_i \neq W'_i$ and $W_j \neq W'_j$.*

The function $g_s$ depends on the selection of some $i, j \in \{1, \dots, k\}$ and thus it must be shown that $g_s$ is well-defined. In the upcoming proof $\mathcal{W}$ and $\mathcal{W}'$ differ in two partitioning sets. The indices $i$ and $j$ basically identify these two partitioning sets.

**Lemma 3.2.27**
*$g_s$ is well-defined.*

*Proof*
First notice that $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and therefore the selection of $i$ and $j$ always is possible. It is left to show that $g_s$ does not depend on the selection of $i, j \in \{1, \dots, k\}$.
Let $i, j \in \{1, \dots, k\}$ with $W_i \neq W'_i$ and $W_j \neq W'_j$. The only other possible selection is $i' = j$ and $j' = i$. Then

$$\begin{aligned} g_s(\mathcal{W}, \mathcal{W}') &= P(W_i) - P(W'_i) + P(W_j) - P(W'_j) \\ &= P(W_{j'}) - P(W'_{j'}) + P(W_{i'}) - P(W'_{i'}) \\ &= P(W_{i'}) - P(W'_{i'}) + P(W_{j'}) - P(W'_{j'}) \\ &= g_s(\mathcal{W}, \mathcal{W}'). \end{aligned}$$

□

The connection between $g_s$ and $s_p$ follows directly.

**Lemma 3.2.28**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinSumPP, $i, j \in \{1, \dots, k\}$ and $\mathcal{W} = \{W_1, \dots, W_k\}$ and $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ two feasible solutions for $\mathcal{I}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$. Then*

$$s_p(\mathcal{W}) - s_p(\mathcal{W}') = g_s(\mathcal{W}, \mathcal{W}')$$

*and therefore*

$$s_p(\mathcal{W}) > s_p(\mathcal{W}') \Leftrightarrow 0 < g_s(\mathcal{W}, \mathcal{W}'), \tag{3.2.4}$$
$$s_p(\mathcal{W}) = s_p(\mathcal{W}') \Leftrightarrow 0 = g_s(\mathcal{W}, \mathcal{W}') \tag{3.2.5}$$

*and*

$$s_p(\mathcal{W}) < s_p(\mathcal{W}') \Leftrightarrow 0 > g_s(\mathcal{W}, \mathcal{W}') \tag{3.2.6}$$

*hold.*

*Proof*
Let $i, j \in \{1, \dots, k\}$ such that $W_i \neq W'_i$ and $W_j \neq W'_j$. Then it is

$$g_s(\mathcal{W}, \mathcal{W}') = P(W_i) - P(W'_i) + P(W_j) - P(W'_j).$$

Further, $W_l \in \mathcal{W} \cap \mathcal{W}'$ for all $l \in \{1, \dots, k\} \backslash \{i, j\}$. Therefore,

$$\begin{aligned} s_p(\mathcal{W}) - s_p(\mathcal{W}') &= \sum_{l=1}^{k} P(W_l) - \sum_{l=1}^{k} P(W'_l) \\ &= P(W_i) - P(W'_i) + P(W_j) - P(W'_j) \\ &= g_s(\mathcal{W}, \mathcal{W}'). \end{aligned}$$

This proves the claim. □

This means the partition $\mathcal{W}'$ is „better“ than partition $\mathcal{W}$ (i.e. the sum subset value is smaller) if $g_s(\mathcal{W}, \mathcal{W}')$ is positive.

In the following the two Algorithms 3.2.29 and 3.2.30 are presented. The idea is to start with some partition and then reduce the sum subset value by moving or switching elements. In Algorithm 3.2.29 the move or switch which is found first is made, whereas in Algorithm 3.2.30 the move or switch which reduces the target function the most is made.

Note that in the algorithms presented in this section often the `continue` command is used. The technical meaning of this command is explained as follows. `continue`, which can only be called in loops, skips the rest of the code in the loop and directly goes back to the beginning of the loop. In nested loops a parameter which specifies the levels of loops to go back can be passed. Thus, in two nested loops `continue 1` goes to the beginning of the inner and `continue 2` to the beginning of the outer loop.

**Algorithm 3.2.29** MinSumPP move/exchange

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinSumPP
- Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W'_1, \ldots, W'_k\}$ of $V$ with $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$

```
1: $\mathcal{W}' \leftarrow \mathcal{W}$
2: loop
3:     for all $(v, j) \in M(\mathcal{W}')$ do
4:         if $0 < g_s(\mathcal{W}', \mathcal{W}'_{(v,j)})$ then
5:             $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
6:             continue 2
7:         end if
8:     end for
9:     for all $(v, w) \in S(\mathcal{W}')$ do
10:        if $0 < g_s(\mathcal{W}', \mathcal{W}'_{(v,w)})$ then
11:            $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,w)}$
12:            continue 2
13:        end if
14:    end for
15:    return $\mathcal{W}'$
16: end loop
```

The following theorem shows that Algorithms 3.2.29 and 3.2.30 terminate. Further, both algorithms produce a feasible solution which is at least as good as the input partition.

**Algorithm 3.2.30** MinSumPP maximal move/exchange

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinSumPP
- Partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ of $V$ with $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$

```
1: $\mathcal{W}' \leftarrow \mathcal{W}$
2: loop
3:     $m \leftarrow \max_{(v,j) \in M(\mathcal{W}')} g_s(\mathcal{W}', \mathcal{W}'_{(v,j)})$
4:     $s \leftarrow \max_{(v,w) \in S(\mathcal{W}')} g_s(\mathcal{W}', \mathcal{W}'_{(v,w)})$
5:     if $s \leq m$ and $0 < m$ then
6:         // Move is better than switch and reduces target function
7:         $(v, j) \in \operatorname{argmax}_{(v,j) \in M(\mathcal{W}')} g_s(\mathcal{W}', \mathcal{W}'_{(v,j)})$
8:         $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
9:         continue 1
10:    else if $m \leq s$ and $0 < s$ then
11:        // Switch is better than move and reduces target function
12:        $(v, w) \in \operatorname{argmax}_{(v,w) \in S(\mathcal{W}')} g_s(\mathcal{W}', \mathcal{W}'_{(v,w)})$
13:        $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,w)}$
14:        continue 1
15:    end if
16:    return $\mathcal{W}'$
17: end loop
```

**Theorem 3.2.31**
*Algorithms 3.2.29 and 3.2.30 terminate after a finite number of steps for an instance $\mathcal{I} = (V, p, k)$ of MinSumPP and partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ as input. The output $\mathcal{W}' = \{W'_1, \ldots, W'_k\}$ is a partition of $V$ with $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$.*

*Proof*
First $\mathcal{W}$ is assigned to $\mathcal{W}'$. Then in each cycle $\mathcal{W}'$ either becomes $\mathcal{W}'_{(v,j)}$ or $\mathcal{W}'_{(v,w)}$, which both are partitions of $V$ since $\mathcal{W}'$ is a partition of $V$. Thus, the returned set also is a partition of $V$.
With Lemma 3.2.28 Equation (3.2.4) („$\Leftarrow$“) in each cycle either the sum subset value of $\mathcal{W}'$ decreases or the current partition is returned. Thus, $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$.
Since there is only a finite number of partitions of $V$ and the sum subset value of $\mathcal{W}'$ always decreases, there is a point where no further descent is possible. With Lemma 3.2.28 Equation (3.2.4) („$\Rightarrow$“) at this point the algorithm terminates. □

## MinMaxPP

For MinMaxPP a second criterion is introduced to compare solutions. First of all, a partition $\mathcal{W}'$ is „better“ than partition $\mathcal{W}$ if $m_p(\mathcal{W}') < m_p(\mathcal{W})$. However, if $m_p(\mathcal{W}') = m_p(\mathcal{W})$, also the number of partitioning sets $W$ (or $W'$) with $P(W) = m_p(\mathcal{W})$ (or $P(W') = m_p(\mathcal{W}')$) is crucial. Often the maximum subset value cannot be reduced by moves or switches because there are three or more of these sets. Thus, secondly a partition where this number is smaller should be preferred. To put this in a formal context the following definition is given.

**Definition 3.2.32**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP and $\mathcal{W} = \{W_1, \ldots, W_k\}$ a partition of $V$. A partitioning set $W_i$ is called* maxset *of $\mathcal{W}$ if and only if*

$$P(W_i) = m_p(\mathcal{W}).$$

*The set*

$$I_{\mathcal{W}} := \{i \in \{1, \ldots, k\} : P(W_i) = m_p(\mathcal{W})\}$$

*is called the set of* maxset indices *of $\mathcal{W}$.*

**Remark 3.2.33**
Note that due to the definition of $m_p(\mathcal{W})$ always $|I_{\mathcal{W}}| \geq 1$ holds.

The following lemma is used in the upcoming proofs and is just stated here to save some work later.

**Lemma 3.2.34**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP and $\mathcal{W} = \{W_1, \dots, W_k\}$ and $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ two feasible solutions for $\mathcal{I}$ with $m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and $P(W_i) \geq P(W'_i)$ for all $i \in \{1, \dots, k\}$. Then $I_{\mathcal{W}'} \subseteq I_{\mathcal{W}}$.*

*Proof*
Let $i \in I_{\mathcal{W}'}$, then

$$m_p(\mathcal{W}) = m_p(\mathcal{W}') = P(W'_i) \leq P(W_i) \leq m_p(\mathcal{W})$$

and thus $P(W_i) = m_p(\mathcal{W})$. This means $i \in I_{\mathcal{W}}$. □

Like for MinSumPP a function $g_m$ which compares two partitions of $V$ can be defined. This function is positive if partition $\mathcal{W}'$ should be preferred over partition $\mathcal{W}$, it is 0 if both partitions are considered equal and otherwise it is $-1$. In this context $-1$ is just some negative number and has no further interpretation. To prefer a partition in this regard means that either the maximum subset value is smaller or the maximum subset value is equal and the number of maxsets is smaller. Thus, a partition $\mathcal{W}'$ can only be preferred over a partition $\mathcal{W}$ if it does not have the same maxsets, i.e. for some $i \in I_{\mathcal{W}}$ it must be $W_i \neq W'_i$. This is considered in the following definition. Besides the two partitions $g_m$ has another parameter $i \in I_{\mathcal{W}}$.

**Definition 3.2.35**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, $\mathcal{W} = \{W_1, \dots, W_k\}$ and $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ two feasible solutions for $\mathcal{I}$ and $i \in I_{\mathcal{W}}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and $W_i \neq W'_i$. Define*

$$g_m(i, \mathcal{W}, \mathcal{W}') := \begin{cases} -1 & \text{if } A \\ 0 & \text{if } B \\ 2P(W_i) - P(W'_i) - P(W'_j) & \text{if } C \end{cases},$$

*where $j \in \{1, \ldots, k\} \backslash \{i\}$ such that $W_j \neq W'_j$ and*

$$\begin{aligned} A = {} & P(W_i) < P(W'_i) \vee \\ & P(W_i) < P(W'_j) \vee \\ & (j \notin I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) = P(W'_j)), \end{aligned}$$

$$\begin{aligned} B = {} & (j \in I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) = P(W'_j)) \vee \\ & (j \notin I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\ & (j \notin I_{\mathcal{W}} \wedge P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)) \end{aligned}$$

*and*

$$\begin{aligned} C = {} & (P(W_i) > P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\ & (j \in I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\ & (j \in I_{\mathcal{W}} \wedge P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)) \end{aligned}$$

*are three logical expressions.*

**Remark 3.2.36**
If

$$\begin{aligned} & (P(W_i) > P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\ & (j \in I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\ & (j \in I_{\mathcal{W}} \wedge P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)) \end{aligned}$$

is `true` in Definition 3.2.35, then $2P(W_i) - P(W'_i) - P(W'_j) > 0$. Thus, it is $g_m(i, \mathcal{W}, \mathcal{W}') > 0$ if and only if $C$ is `true`.

The function $g_m$ depends on the selection of some $j \in \{1, \ldots, k\} \backslash \{i\}$ and thus it must be shown that $g_m$ is well-defined. Further, it has to be verified that exactly one of the expressions $A$, $B$ or $C$ is `true`.

**Lemma 3.2.37**
*$g_m$ is well-defined.*

*Proof*
That exactly one of the expressions $A$, $B$ or $C$ is `true` can be seen in a disjunct lineup like this:

- $P(W_i) < P(W'_i) \vee P(W_i) < P(W'_j)$: A
- $\overline{P(W_i) < P(W'_i) \vee P(W_i) < P(W'_j)}$
  - $P(W_i) > P(W'_i) \wedge P(W_i) > P(W'_j)$: C
  - $P(W_i) = P(W'_i) \wedge P(W_i) = P(W'_j)$
    - $j \in I_{\mathcal{W}}$: B
    - $j \notin I_{\mathcal{W}}$: A
  - $P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j)$
    - $j \in I_{\mathcal{W}}$: C
    - $j \notin I_{\mathcal{W}}$: B
  - $P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)$
    - $j \in I_{\mathcal{W}}$: C
    - $j \notin I_{\mathcal{W}}$: B

Since $j \in \{1, \ldots, k\} \backslash \{i\}$, $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and $W_i \neq W'_i$, the selection of $j$ always is possible and unique. □

The following lemmas characterize the connection between $g_m$ and the maximum subset value and the number of maxsets of two partitions. For two partitions with equal maximum subset value the partition with the smaller number of maxsets should be preferred. Thus, mathematically speaking, $\mathcal{W}'$ is „better“ than $\mathcal{W}$ if

$$(m_p(\mathcal{W}) \geq m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| > |I_{\mathcal{W}'}|) \vee m_p(\mathcal{W}) > m_p(\mathcal{W}').$$

**Lemma 3.2.38**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, $\mathcal{W} = \{W_1, \ldots, W_k\}$ and $\mathcal{W}' = \{W'_1, \ldots, W'_k\}$ two feasible solutions for $\mathcal{I}$ and $i \in I_{\mathcal{W}}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and $W_i \neq W'_i$. Then*

$$(m_p(\mathcal{W}) \geq m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| > |I_{\mathcal{W}'}|) \vee m_p(\mathcal{W}) > m_p(\mathcal{W}') \Leftrightarrow 0 < g_m(i, \mathcal{W}, \mathcal{W}')$$

*holds.*

*Proof*
Let $j \in \{1,\dots,k\}\backslash\{i\}$ such that $W_j \neq W'_j$. It is to show that

$$(m_p(\mathcal{W}) \geq m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| > |I_{\mathcal{W}'}|) \vee m_p(\mathcal{W}) > m_p(\mathcal{W}')$$

if and only if

$$\begin{aligned}&(P(W_i) > P(W'_i) \wedge P(W_i) > P(W'_j))\vee\\ &(j \in I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j))\vee\\ &(j \in I_{\mathcal{W}} \wedge P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)).\end{aligned}$$

It is

$$m_p(\mathcal{W}) \left\{\begin{matrix} > \\ \geq \end{matrix}\right\} m_p(\mathcal{W}') \Rightarrow P(W_i) \left\{\begin{matrix} > \\ \geq \end{matrix}\right\} P(W'_i) \wedge P(W_i) \left\{\begin{matrix} > \\ \geq \end{matrix}\right\} P(W'_j) \tag{3.2.7}$$

since

$$m_p(\mathcal{W}) \left\{\begin{matrix} > \\ \geq \end{matrix}\right\} m_p(\mathcal{W}') \Rightarrow P(W_i) = m_p(\mathcal{W}) \left\{\begin{matrix} > \\ \geq \end{matrix}\right\} m_p(\mathcal{W}') \geq \begin{cases} P(W'_i) \\ P(W'_j) \end{cases}$$

holds.

- Case $I_{\mathcal{W}} = \{i\}$:
  Since $|I_{\mathcal{W}'}| \geq 1$ and $j \notin I_{\mathcal{W}}$, it remains to show

  $$m_p(\mathcal{W}) > m_p(\mathcal{W}') \Leftrightarrow P(W_i) > P(W'_i) \wedge P(W_i) > P(W'_j).$$

  „$\Rightarrow$“: Follows with Equation (3.2.7).

  „$\Leftarrow$“: For all $l \in \{1,\dots,k\}\backslash\{i,j\}$ it is $W_l = W'_l$ and therefore $m_p(\mathcal{W}) = P(W_i) > P(W_l) = P(W'_l)$. Further,

  $$m_p(\mathcal{W}) = P(W_i) > \begin{cases} P(W'_i) \\ P(W'_j) \end{cases}$$

  and thus $m_p(\mathcal{W}) > m_p(\mathcal{W}')$.

- Case $|I_{\mathcal{W}}| \geq 2$ and $j \in I_{\mathcal{W}}$:
  It is $P(W_i) = P(W_j)$.

„$\Rightarrow$“: If $m_p(\mathcal{W}) > m_p(\mathcal{W}')$, then $P(W_i) > P(W_i')$ and $P(W_i) > P(W_j')$ follows directly with Equation (3.2.7).
If $m_p(\mathcal{W}) \geq m_p(\mathcal{W}')$ and $|I_\mathcal{W}| > |I_{\mathcal{W}'}|$, then $P(W_i) \geq P(W_i')$ and $P(W_i) \geq P(W_j')$ also with Equation (3.2.7). Assumed it was $P(W_i) = P(W_i')$ and $P(W_i) = P(W_j')$. Because with $W_l = W_l'$ for all $l \in \{1,\ldots,k\}\backslash\{i,j\}$ then it also was $I_\mathcal{W} = I_{\mathcal{W}'}$. This is a contradiction. Thus, $P(W_i) > P(W_i')$ or $P(W_i) > P(W_j')$.

„$\Leftarrow$“: Let without loss of generality $P(W_i) > P(W_i')$ and $P(W_j) = P(W_i) \geq P(W_j')$ (since $i, j \in I_\mathcal{W}$ both indices can be exchanged). Then either $m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and thus $i \notin I_{\mathcal{W}'}$ and $I_{\mathcal{W}'} \subseteq I_\mathcal{W}\backslash\{i\}$ with Lemma 3.2.34 or $m_p(\mathcal{W}) > m_p(\mathcal{W}')$.

- Case $|I_\mathcal{W}| \geq 2$ and $j \notin I_\mathcal{W}$:
There exists $l \in \{1,\ldots,k\}\backslash\{i,j\}$ with $l \in I_\mathcal{W}$ and thus $m_p(\mathcal{W}) = m_p(\mathcal{W}')$. Since $W_l = W_l'$ for all $l \in \{1,\ldots,k\}\backslash\{i,j\}$, it follows $I_\mathcal{W}\backslash\{i\} = I_{\mathcal{W}'}\backslash\{i,j\}$. It remains to show

$$|I_\mathcal{W}| > |I_{\mathcal{W}'}| \Leftrightarrow P(W_i) > P(W_i') \wedge P(W_i) > P(W_j').$$

„$\Rightarrow$“: Assumed $i \in I_{\mathcal{W}'}$ or $j \in I_{\mathcal{W}'}$, then $|I_\mathcal{W}| = |I_\mathcal{W}\backslash\{i\}|+1 = |I_{\mathcal{W}'}\backslash\{i,j\}|+1 \leq |I_{\mathcal{W}'}|$, which is in contradiction to $|I_\mathcal{W}| > |I_{\mathcal{W}'}|$. Therefore, $P(W_i') < m_p(\mathcal{W}') = m_p(\mathcal{W}) = P(W_i)$ and $P(W_j') < m_p(\mathcal{W}') = m_p(\mathcal{W}) = P(W_i)$.

„$\Leftarrow$“: It is $P(W_i') < P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and $P(W_j') < P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$. Thus, $i, j \notin I_{\mathcal{W}'}$ and therefore $|I_\mathcal{W}| = |I_\mathcal{W}\backslash\{i\}| + 1 = |I_{\mathcal{W}'}\backslash\{i,j\}| + 1 = |I_{\mathcal{W}'}| + 1$.

□

Two partitions should only be considered as equal if the maximum subset value and the number of maxsets are equal. The connection to $g_m$ is given in the following lemma.

**Lemma 3.2.39**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, $\mathcal{W} = \{W_1,\ldots,W_k\}$ and $\mathcal{W}' = \{W_1',\ldots,W_k'\}$ two feasible solutions for $\mathcal{I}$ and $i \in I_\mathcal{W}$, where $|\mathcal{W} \cap \mathcal{W}'| = k-2$ and $W_i \neq W_i'$.*

*Then*

$$m_p(\mathcal{W}) = m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| = |I_{\mathcal{W}'}| \Leftrightarrow 0 = g_m(i, \mathcal{W}, \mathcal{W}')$$

*holds.*

*Proof*
Let $j \in \{1, \dots, k\} \backslash \{i\}$ such that $W_j \neq W'_j$. Like in the proof of Lemma 3.2.38 it is to show

$$m_p(\mathcal{W}) = m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| = |I_{\mathcal{W}'}|$$

if and only if

$$\begin{aligned}
&(j \in I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) = P(W'_j)) \vee \\
&(j \notin I_{\mathcal{W}} \wedge P(W_i) = P(W'_i) \wedge P(W_i) > P(W'_j)) \vee \\
&(j \notin I_{\mathcal{W}} \wedge P(W_i) > P(W'_i) \wedge P(W_i) = P(W'_j)).
\end{aligned}$$

„$\Rightarrow$“: The inequalities

$$P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}') \geq \begin{cases} P(W'_i) \\ P(W'_j) \end{cases}$$

hold. Further, since $W_l = W'_l$ for all $l \in \{1, \dots, k\} \backslash \{i, j\}$ and $m_p(\mathcal{W}) = m_p(\mathcal{W}')$, it is $I_{\mathcal{W}} \backslash \{i, j\} = I_{\mathcal{W}'} \backslash \{i, j\}$.
If $j \in I_{\mathcal{W}}$, then also $P(W_j) = P(W_i) \geq P(W'_j)$ and thus with Lemma 3.2.34 $I_{\mathcal{W}'} \subseteq I_{\mathcal{W}}$. Since $|I_{\mathcal{W}}| = |I_{\mathcal{W}'}|$, this means $I_{\mathcal{W}'} = I_{\mathcal{W}}$ and therefore

$$P(W'_i) = P(W'_j) = m_p(\mathcal{W}') = m_p(\mathcal{W}) = P(W_i) = P(W_j).$$

If $j \notin I_{\mathcal{W}}$, then $P(W_i) \neq P(W'_i)$ or $P(W_i) \neq P(W'_j)$. Otherwise $P(W'_i) = P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and $P(W'_j) = P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and then $I_{\mathcal{W}'} = I_{\mathcal{W}} \cup \{j\}$, which is in contradiction to $|I_{\mathcal{W}}| = |I_{\mathcal{W}'}|$. Further, $P(W_i) = P(W'_i)$ or $P(W_i) = P(W'_j)$. Otherwise $P(W'_i) < P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and $P(W'_j) < P(W_i) = m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and then $I_{\mathcal{W}'} = I_{\mathcal{W}} \backslash \{i\}$, which also is in contradiction to $|I_{\mathcal{W}}| = |I_{\mathcal{W}'}|$.

„$\Leftarrow$“: With $P(W_i) = P(W_i') \wedge P(W_i) = P(W_j')$, $P(W_i) = P(W_i') \wedge P(W_i) > P(W_j')$ or $P(W_i) > P(W_i') \wedge P(W_i) = P(W_j')$ it follows directly $m_p(\mathcal{W}) = m_p(\mathcal{W}')$ and thus like before $I_{\mathcal{W}} \backslash \{i, j\} = I_{\mathcal{W}'} \backslash \{i, j\}$.
If $j \in I_{\mathcal{W}}$, then also $P(W_i) = P(W_i') \wedge P(W_i) = P(W_j')$. This means $i, j \in I_{\mathcal{W}'}$ and thus $I_{\mathcal{W}} = I_{\mathcal{W}'}$.
If $j \notin I_{\mathcal{W}}$, then either $P(W_i) = P(W_i') \wedge P(W_i) > P(W_j')$, and thus $i \in I_{\mathcal{W}'}$ and $j \notin I_{\mathcal{W}'}$, or $P(W_i) > P(W_i') \wedge P(W_i) = P(W_j')$, and thus $i \notin I_{\mathcal{W}'}$ and $j \in I_{\mathcal{W}'}$. In Both cases $|I_{\mathcal{W}}| = |I_{\mathcal{W}'}|$.

□

The following lemma is just given to shorten the proof of Lemma 3.2.41.

**Lemma 3.2.40**
*Let $a, b \in \mathbb{R}$ and $c, d \in \mathbb{N}$. Then*

$$a < b \vee (c < d \wedge a \leq b) \Leftrightarrow \overline{(a \geq b \wedge c > d) \vee a > b} \wedge \overline{a = b \wedge c = d}.$$

*Proof*
The following implications hold:

$$\begin{aligned}
& a < b \vee (c < d \wedge a \leq b) \\
\Leftrightarrow\ & a < b \vee (c \leq d \wedge c \neq d \wedge a \leq b) \\
\Leftrightarrow\ & (a < b \vee c \leq d) \wedge (a < b \vee (a \leq b \wedge c \neq d)) \\
\Leftrightarrow\ & (a < b \vee c \leq d) \wedge (a \leq b \wedge (a \neq b \vee c \neq d)) \\
\Leftrightarrow\ & (a < b \vee c \leq d) \wedge a \leq b) \wedge (a \neq b \vee c \neq d) \\
\Leftrightarrow\ & \overline{(a \geq b \wedge c > d) \vee a > b} \wedge \overline{a = b \wedge c = d}
\end{aligned}$$

□

**Lemma 3.2.41**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, $\mathcal{W} = \{W_1, \dots, W_k\}$ and $\mathcal{W}' = \{W_1', \dots, W_k'\}$ two feasible solutions for $\mathcal{I}$ and $i \in I_{\mathcal{W}}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and $W_i \neq W_i'$. Then*

$$m_p(\mathcal{W}) < m_p(\mathcal{W}') \vee (|I_{\mathcal{W}}| < |I_{\mathcal{W}'}| \wedge m_p(\mathcal{W}) \leq m_p(\mathcal{W}')) \Leftrightarrow 0 > g_m(i, \mathcal{W}, \mathcal{W}')$$

*holds.*

*Proof*
With Lemma 3.2.40 it is

$$m_p(\mathcal{W}) < m_p(\mathcal{W}') \vee (|I_{\mathcal{W}}| < |I_{\mathcal{W}'}| \wedge m_p(\mathcal{W}) \leq m_p(\mathcal{W}'))$$

if and only if

$$\begin{aligned}&\overline{(m_p(\mathcal{W}) \geq m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| > |I_{\mathcal{W}'}|) \vee m_p(\mathcal{W}) > m_p(\mathcal{W}')} \wedge \\ &\overline{m_p(\mathcal{W}) = m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| = |I_{\mathcal{W}'}|}.\end{aligned}$$

The claim follows with Lemmas 3.2.38 and 3.2.39. □

The maximum subset value or the number of maxsets just are decreased by a move or switch if a maxset is changed. Thus, in the following definition sets of moves or switches which affect a certain set of partitioning sets are defined.

**Definition 3.2.42**
*Let $V$ be a finite set, $\mathcal{W} = \{W_1, \ldots, W_k\}$ a partition of $V$ with $k \in \mathbb{N}$ and $I \subseteq \{1, \ldots, k\}$. Define*

$$M(\mathcal{W}; I) := \{(v, j, i) \in M(\mathcal{W}) \times I : v \in W_i \vee j = i\}$$

*and*

$$S(\mathcal{W}; I) := \{(v, w, i) \in S(\mathcal{W}) \times I : v \in W_i \vee w \in W_i\}.$$

Using this definition only the moves from $M(\mathcal{W}; I_{\mathcal{W}})$ or switches from $S(\mathcal{W}; I_{\mathcal{W}})$ can reduce the maxium subset value or the number of maxsets of a partition $\mathcal{W}$.

As for MinSumPP the following algorithms represent the approach of moving one element from one partitioning set to another or switching two elements. In Algorithm 3.2.43 the first move or switch, where $g_m$ is positive, is made, whereas in Algorithm 3.2.44 the move or switch, where $g_m$ is maximal, is made. Note that the value of $g_m$ does not really reflect the actual reduction of the target function. The function $g_m$ is even positive if the maximum subset value stays the same but the number of maxsets reduces. In fact, in this case $g_m$ compares the subset values of the two changed partitioning sets with the old maximum subset value. However, then a large $g_m$ implies that the subset values of these two partitioning sets are small and therefore maximizing $g_m$ is eligible.

**Algorithm 3.2.43** MinMaxPP move/exchange

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinMaxPP
- Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W_1', \ldots, W_k'\}$ of $V$ with $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$

```
 1: $\mathcal{W}' \leftarrow \mathcal{W}$
 2: loop
 3:     for all $(v, j, i) \in M(\mathcal{W}'; I_{\mathcal{W}'})$ do
 4:         if $0 < g_m(i, \mathcal{W}', \mathcal{W}'_{(v,j)})$ then
 5:             $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
 6:             continue 2
 7:         end if
 8:     end for
 9:     for all $(v, w, i) \in S(\mathcal{W}'; I_{\mathcal{W}'})$ do
10:         if $0 < g_m(i, \mathcal{W}', \mathcal{W}'_{(v,w)})$ then
11:             $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,w)}$
12:             continue 2
13:         end if
14:     end forreturn $\mathcal{W}'$
15: end loop
```

The following theorem shows that Algorithms 3.2.43 and 3.2.44 terminate. Further, both algorithms produce a feasible solution which is at least as good as the input partition.

**Theorem 3.2.45**
*Algorithms 3.2.43 and 3.2.44 terminate after a finite number of steps for an instance $\mathcal{I} = (V, p, k)$ of MinMaxPP and partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$ as input. The output $\mathcal{W}' = \{W_1', \ldots, W_k'\}$ is a partition of $V$ with $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$.*

*Proof*
First $\mathcal{W}$ is assigned to $\mathcal{W}'$. Then in each cycle $\mathcal{W}'$ either becomes $\mathcal{W}'_{(v,j)}$ or $\mathcal{W}'_{(v,w)}$, which both are partitions of $V$ since $\mathcal{W}'$ is a partition of $V$. Thus, the returned set is also a partition of $V$.
With Lemma 3.2.38 („$\Leftarrow$“) in each cycle either the maximum subset value of $\mathcal{W}'$ decreases or the size of $I_{\mathcal{W}'}$ decreases while the maximum subset value is

**Algorithm 3.2.44** MinMaxPP maximal move/exchange

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinMaxPP
- Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W'_1, \ldots, W'_k\}$ of $V$ with $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$

```
1:  $\mathcal{W}' \leftarrow \mathcal{W}$
2:  **loop**
3:      $m \leftarrow \max_{(v,j,i) \in M(\mathcal{W}'; I_{\mathcal{W}'})} g_m(i, \mathcal{W}', \mathcal{W}'_{(v,j)})$
4:      $s \leftarrow \max_{(v,w,i) \in S(\mathcal{W}'; I_{\mathcal{W}'})} g_m(i, \mathcal{W}', \mathcal{W}'_{(v,w)})$
5:      **if** $s \leq m$ **and** $0 < m$ **then**
6:          // Move is better than switch and reduces target
7:          $(v, j, i) \in \operatorname{argmax}_{(v,j,i) \in M(\mathcal{W}'; I_{\mathcal{W}'})} g_m(i, \mathcal{W}', \mathcal{W}'_{(v,j)})$
8:          $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
9:          **continue** 1
10:     **else if** $m \leq s$ **and** $0 < s$ **then**
11:         // Switch is better than move and reduces target
12:         $(v, w, i) \in \operatorname{argmax}_{(v,w,i) \in S(\mathcal{W}'; I_{\mathcal{W}'})} g_m(i, \mathcal{W}', \mathcal{W}'_{(v,w)})$
13:         $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,w)}$
14:         **continue** 1
15:     **end if**
16:     **return** $\mathcal{W}'$
17: **end loop**
```

unchanged or the current partition is returned. Thus, $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$.
If the maximum subset value is unchanged in one cycle, the size of $I_{\mathcal{W}'}$ decreases. Since $|I_{\mathcal{W}'}|$ always has to be larger than 0, after a finite number of steps the maximum subset value has to decrease. Further, since there is only a finite number of partitions of $V$ and the maximum subset value of $\mathcal{W}'$ decreases, there is a point where no further descent is possible and the size of $I_{\mathcal{W}'}$ cannot be reduced. With Lemma 3.2.38 („$\Rightarrow$“) the algorithm terminates at this point. □

**Expand search regions**

Algorithms 3.2.29, 3.2.30, 3.2.43 and 3.2.44 just move or switch elements if a real improvement can be gained. To reduce the probability to get stuck in a local optimum the search region can be expanded in the cases where no real improvement can be found. To do this a search tree is created. In this tree all partitions are stored which are of equal preference as the current partition. If a real improvement can be achieved via one of these partitions, the according partition is used as next partition and the search tree is deleted.
The algorithms in this section are rather of conceptual purpose. They are not implemented and tested as efficient programming of a search tree is not in the focus of this work. The main reason that a search tree approach works is that the equality of preference gets entailed in the tree. This is proven in the following lemma.

**Lemma 3.2.46**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinSumPP, $\mathcal{W} = \{W_1, \ldots, W_k\}$, $\mathcal{W}' = \{W_1', \ldots, W_k'\}$ and $\mathcal{W}'' = \{W_1'', \ldots, W_k''\}$ three feasible solutions for $\mathcal{I}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$ and $|\mathcal{W}' \cap \mathcal{W}''| = k - 2$. Then*

$$g_s(\mathcal{W}, \mathcal{W}') = g_s(\mathcal{W}', \mathcal{W}'') = 0 \Rightarrow s_p(\mathcal{W}) = s_p(\mathcal{W}'')$$

*holds.*

*Proof*
Since $g_s(\mathcal{W}, \mathcal{W}') = 0$ and $g_s(\mathcal{W}', \mathcal{W}'') = 0$, with Lemma 3.2.28 it follows

$$s_p(\mathcal{W}) = s_p(\mathcal{W}') \quad \text{and} \quad s_p(\mathcal{W}') = s_p(\mathcal{W}'').$$

□

In Algorithm 3.2.47 the search tree is stored in the set $\mathcal{X}$ and partitions are only added if they are of equal preference. The search tree from the last iteration is saved in $\mathcal{X}_o$. The algorithm terminates if no real improvement and no further partitions of equal preference can be found.

**Algorithm 3.2.47** MinSumPP global move

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinSumPP
- Partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ of $V$ with $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$

```
1:  $\mathcal{W}' \leftarrow \mathcal{W}$
2:  loop
3:      for all $(v, j) \in M(\mathcal{W}')$ do
4:          if $0 < g_s(\mathcal{W}', \mathcal{W}'_{(v,j)})$ then
5:              $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
6:              continue 2
7:          end if
8:      end for
9:      $\mathcal{X}_o \leftarrow \emptyset$
10:     $\mathcal{X} \leftarrow \{\mathcal{W}'\}$
11:     while $\mathcal{X} \neq \mathcal{X}_o$ do
12:         for all $\mathcal{V} \in \mathcal{X}, (v, j) \in M(\mathcal{V})$ do
13:             if $\mathcal{V}_{(v,j)} \notin \mathcal{X}$ then
14:                 if $0 < g_s(\mathcal{V}, \mathcal{V}_{(v,j)})$ then
15:                     $\mathcal{W}' \leftarrow \mathcal{V}_{(v,j)}$
16:                     continue 3
17:                 else if $0 = g_s(\mathcal{V}, \mathcal{V}_{(v,j)})$ then
18:                     $\mathcal{X}_o \leftarrow \mathcal{X}$
19:                     $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathcal{V}_{(v,j)}\}$
20:                 end if
21:             end if
22:         end for
23:     end while
24:     return $\mathcal{W}'$
25: end loop
```

**Theorem 3.2.48**

*Algorithm 3.2.47 terminates after a finite number of steps for an instance $\mathcal{I} = (V, p, k)$ of MinSumPP and partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ as input. The output is a partition $\mathcal{W}' = \{W'_1, \dots, W'_k\}$ of $V$ with $s_p(\mathcal{W}') \leq s_p(\mathcal{W})$.*

*Proof*
First $\mathcal{W}$ is assigned to $\mathcal{W}'$, which is a partition of $V$. Since all involved sets are derived by possible moves, they all are partitions of $V$.
If the `if` query in line 4 is `true`, then with Lemma 3.2.28 Equation (3.2.4) the sum subset value of $\mathcal{W}'$ decreases. If the `if` query is `false`, a new search tree $\mathcal{X}$ which only contains $\mathcal{W}'$ is generated. With Lemma 3.2.46 for every added partition $\mathcal{V}_{(v,j)}$ it is $s_p(\mathcal{V}_{(v,j)}) = s_p(\mathcal{W}')$. The `while` loop in line 11 terminates since never elements are removed from $\mathcal{X}$ and there is just a finite number of partitions of $V$.
If the `if` query in line 14 is `true`, then the sum subset value of $\mathcal{W}'$ decreases since $s_p(\mathcal{V}) = s_p(\mathcal{W}')$ for all $\mathcal{V} \in \mathcal{X}$ and thus $s_p(\mathcal{V}_{(v,j)}) < s_p(\mathcal{V}) = s_p(\mathcal{W}')$. Otherwise the current $\mathcal{W}'$ is returned. Thus, in each loop of line 2 either the sum subset value of $\mathcal{W}'$ decreases or the current $\mathcal{W}'$ is returned. Since there is only a finite number of partitions, with Equation (3.2.4) in Lemma 3.2.28 the claim is proven. □

**Remark 3.2.49**
Algorithm 3.2.47 is an extension of Algorithm 3.2.29 without switching elements. Algorithm 3.2.30 can be modified in the same way to expand the search region and also switching elements can be added easily.

The following lemma shows that the equality of preference also gets entailed in the search tree for MinMaxPP.

**Lemma 3.2.50**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, $\mathcal{W} = \{W_1, \ldots, W_k\}$, $\mathcal{W}' = \{W'_1, \ldots, W'_k\}$ and $\mathcal{W}'' = \{W''_1, \ldots, W''_k\}$ three feasible solutions for $\mathcal{I}$, where $|\mathcal{W} \cap \mathcal{W}'| = k - 2$, $|\mathcal{W}' \cap \mathcal{W}''| = k - 2$, $i \in I_{\mathcal{W}}$ and $i' \in I_{\mathcal{W}'}$. If*

$$g_m(i, \mathcal{W}, \mathcal{W}') = g_m(i', \mathcal{W}', \mathcal{W}'') = 0,$$

*then*

$$m_p(\mathcal{W}) = m_p(\mathcal{W}'') \wedge |I_{\mathcal{W}}| = |I_{\mathcal{W}''}|$$

*holds.*

*Proof*

Since $g_m(i, \mathcal{W}, \mathcal{W}') = 0$ and $g_m(i', \mathcal{W}', \mathcal{W}'') = 0$, with Lemma 3.2.39 it follows

$$m_p(\mathcal{W}) = m_p(\mathcal{W}') \wedge |I_{\mathcal{W}}| = |I_{\mathcal{W}'}| \quad \text{and} \quad m_p(\mathcal{W}') = m_p(\mathcal{W}'') \wedge |I_{\mathcal{W}'}| = |I_{\mathcal{W}''}|.$$

□

**Algorithm 3.2.51** MinMaxPP global move

**Input:**
- Instance $\mathcal{I} = (V, p, k)$ of MinMaxPP
- Partition $\mathcal{W} = \{W_1, \ldots, W_k\}$ of $V$

**Output:** Partition $\mathcal{W}' = \{W_1', \ldots, W_k'\}$ of $V$ with $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$

```
1: $\mathcal{W}' \leftarrow \mathcal{W}$
2: loop
3:     $\mathcal{W}' \leftarrow \emptyset$
4:     for all $(v, j, i) \in M(\mathcal{W}'; I_{\mathcal{W}'})$ do
5:         if $0 < g_m(i, \mathcal{W}', \mathcal{W}'_{(v,j)})$ then
6:             $\mathcal{W}' \leftarrow \mathcal{W}'_{(v,j)}$
7:             continue 2
8:         end if
9:     end for
10:    $\mathcal{X}_o \leftarrow \emptyset$
11:    $\mathcal{X} \leftarrow \{\mathcal{W}'\}$
12:    while $\mathcal{X} \neq \mathcal{X}_o$ do
13:        for all $\mathcal{V} \in \mathcal{X}, (v, j, i) \in M(\mathcal{V}; I_{\mathcal{V}})$ do
14:            if $\mathcal{V}_{(v,j)} \notin \mathcal{X}$ then
15:                if $0 < g_m(i, \mathcal{V}, \mathcal{V}_{(v,j)})$ then
16:                    $\mathcal{W}' \leftarrow \mathcal{V}_{(v,j)}$
17:                    break 3
18:                else if $0 = g_m(i, \mathcal{V}, \mathcal{V}_{(v,j)})$ then
19:                    $\mathcal{X}_o \leftarrow \mathcal{X}$
20:                    $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathcal{V}_{(v,j)}\}$
21:                end if
22:            end if
23:        end for
24:    end while
25:    return $\mathcal{W}'$
26: end loop
```

**Theorem 3.2.52**
*Algorithm 3.2.51 terminates after a finite number of steps for an instance $\mathcal{I} = (V, p, k)$ of MinMaxPP and partition $\mathcal{W} = \{W_1, \dots, W_k\}$ of $V$ as input. The output is a partition $\mathcal{W}' = \{W_1', \dots, W_k'\}$ of $V$ with $m_p(\mathcal{W}') \leq m_p(\mathcal{W})$.*

*Proof*
As in the proof of Theorem 3.2.48 the set $\mathcal{W}'$ is a partition of $V$.
If the `if` query in line 5 is `true`, with Lemma 3.2.38 the maximum subset value of $\mathcal{W}'$ decreases or the size of $I_{\mathcal{W}'}$ decreases while the maximum subset value is unchanged. If the `if` query is `false`, a new search tree $\mathcal{X}$ which only contains $\mathcal{W}'$ is generated. With Lemma 3.2.50 for every recursively added partition $\mathcal{V}_{(v,j)}$ it is $m_p(\mathcal{V}_{(v,j)}) = m_p(\mathcal{W}')$ and $|I_{\mathcal{V}_{(v,j)}}| = |I_{\mathcal{W}'}|$. The `while` loop in line 12 terminates since never elements are removed from $\mathcal{X}$ and there is just a finite number of partitions of $V$.
It is $m_p(\mathcal{V}) = m_p(\mathcal{W}')$ and $|I_{\mathcal{V}}| = |I_{\mathcal{W}'}|$ for all $\mathcal{V} \in \mathcal{X}$. Thus, if the `if` query in line 15 is `true`, the maximum subset value decreases or the number of maxsets decreases while the maximum subset value is unchanged since $m_p(\mathcal{V}_{(v,j)}) < m_p(\mathcal{V}) = m_p(\mathcal{W}')$, or $|I_{\mathcal{V}_{(v,j)}}| < |I_{\mathcal{V}}| = |I_{\mathcal{W}'}|$ and $m_p(\mathcal{V}_{(v,j)}) = m_p(\mathcal{V}) = m_p(\mathcal{W}')$. Otherwise the current $\mathcal{W}'$ is returned. Thus, in each loop of line 2 either the maximum subset value of $\mathcal{W}'$ decreases or the size of $I_{\mathcal{W}'}$ decreases while the maximum subset value is unchanged or the current $\mathcal{W}'$ is returned. Since $|I_{\mathcal{W}'}|$ has to be always greater than 0, after a finite number of steps the maximum subset value has to decrease. Thus, as there is only a finite number of partitions, with Lemma 3.2.38 the claim is proven. $\square$

**Remark 3.2.53**
As before, Algorithm 3.2.51 is an extension of Algorithm 3.2.43 without switching elements. Algorithm 3.2.44 can be modified in the same way to expand the search region and also switching elements can be added easily.

### 3.2.2 Approximation

For the case, where $p$ is not monotonically decreasing or $P$ is not monotonically increasing, in Section 3.2 it was demonstrated that $\text{MinSumPP}_{\neq\emptyset}$ and $\text{MinMaxPP}_{\neq\emptyset}$ do not admit polynomial-time approximation algorithms with a polynomial approximation ratio. Thus, in this section $p$ is assumed to be monotonically decreasing and $P$ is assumed to be monotonically increasing. Approximation algorithms for MinSumPP$_{\neq\emptyset}$ and MinMaxPP$_{\neq\emptyset}$ with polynomial approximation ratio are developed. These results complete the approximation analysis for the minimization type partition problems.

### MinSumPP

With Lemma 3.2.5 already the trivial partition is optimal for MinSumPP if $p$ is monotonically decreasing. Thus, in this section an approximation algorithm is only presented for MinSumPP$_{\neq\emptyset}$.
The leading thought for Algorithm 3.2.54 is to distribute the „expensive“ elements of $V$ among the partitioning sets and then fill up the sets with the „cheaper“ ones. For monotonically decreasing $p$ this should reduce the weight of the former and thus produce a good solution. As it is demonstrated later, this approach leads to a multiplicative error of at most $(1 + |V| - k)$.

**Algorithm 3.2.54** MinSumPP$_{\neq\emptyset}$ approximation

**Input:** • Instance $\mathcal{I} = (V, p, k)$ of MinSumPP$_{\neq\emptyset}$

**Output:** Feasible solution $\mathcal{W} = \{W_1, \dots, W_k\}$ for $\mathcal{I}$

```
1: $n \leftarrow |V|$
2: Sort and rename $V = \{v_1, \dots, v_n\}$ such that $P(\{v_1\}) \geq \dots \geq P(\{v_n\})$
3: for $i = 1 \to k$ do
4:     $W_i \leftarrow \emptyset$
5: end for
6: for $j = 1 \to n$ do
7:     $W_{((j-1) \bmod k)+1} \leftarrow W_{((j-1) \bmod k)+1} \cup \{v_j\}$
8: end for
9: $\mathcal{W} \leftarrow \{W_1, \dots, W_k\}$
10: return $\mathcal{W}$
```

First it is shown that Algorithm 3.2.54 terminates and produces a feasible solution for MinSumPP$_{\neq\emptyset}$.

**Lemma 3.2.55**
*With an instance $\mathcal{I} = (V, p, k)$ of MinSumPP$_{\neq\emptyset}$ as input Algorithm 3.2.54 runs in polynomial time in the size of the input, terminates and returns a feasible solution $\mathcal{W} = \{W_1, \dots, W_k\}$ for $\mathcal{I}$.*

*Proof*
Since sorting can be done in polynomial time, Algorithm 3.2.54 runs in polynomial time. It terminates since all involved loop constructs are `for` loops. Further, note that

$$((j-1) \bmod k) + 1 \in \{1, \dots, k\}$$

for $j \in \{1, \dots, |V|\}$. In the `for` loop (line 6) each $v \in V$ is added exactly once to a set $W_i$ with $i \in \{1, \dots, k\}$. Thus, $\mathcal{W}$ forms a partition of $V$. Since $v_i \in W_i$ and $|V| \geq k$, it is $W_i \neq \emptyset$ for $i \in \{1, \dots, k\}$ and $\mathcal{W}$ is a feasible solution for $\mathcal{I}$. □

To prove the approximation ratio a proof by induction is be used. Thus, some elements of $V$ are removed. For the resulting set $\overline{V}$ the induction base case holds and the idea is to show that adding the elements which were removed before does not increase the target value too much. In this context an upper bound for the optimal sum subset value of $\overline{V}$ is needed. This bound is provided by the following lemma.

**Lemma 3.2.56**
*Let $\mathcal{I} = (V, p, k)$ and $\overline{\mathcal{I}} = (\overline{V}, p, k)$ be two instances of MinSumPP$_{\neq\emptyset}$, where $\overline{V} \subseteq V$ and $P$ is monotonically increasing. Further, let $\mathcal{W}^* = \{W_1^*, \dots, W_k^*\}$ be an optimal solution for $\mathcal{I}$ and $\overline{\mathcal{W}}^*$ an optimal solution for $\overline{\mathcal{I}}$. If $P(\{\overline{v}\}) \leq P(\{v\})$ for all $\overline{v} \in \overline{V}$ and $v \in V \backslash \overline{V}$, then*

$$s_p(\overline{\mathcal{W}}^*) \leq s_p(\mathcal{W}^*).$$

*Proof*
Assumed $s_p(\overline{\mathcal{W}}^*) > s_p(\mathcal{W}^*)$. Define $\overline{\mathcal{W}}' := \{\overline{W}'_1, \dots, \overline{W}'_k\}$ with $\overline{W}'_i := W_i^* \cap \overline{V}$. Since $\mathcal{W}^*$ is a partition of $V$ and $\overline{V} \subseteq V$, $\overline{\mathcal{W}}'$ is a partition of $\overline{V}$. Further, $P(\overline{W}'_i) \leq P(W_i)$ because $\overline{W}'_i \subseteq W_i$ for all $i \in \{1, \dots, k\}$.
If there is $i \in \{1, \dots, k\}$ with $\overline{W}'_i = \emptyset$, then $W_i^* \subseteq V \backslash \overline{V}$ and therefore

$$P(\{\overline{v}\}) \leq P(\{w_i^*\})$$

for all $\overline{v} \in \overline{V}$ and $w_i^* \in W_i^* \neq \emptyset$.

Since $|\overline{V}| \geq k$, there exists a $j \in \{1, \ldots, k\}$ with $j \neq i$ and $|\overline{W}'_j| \geq 2$. Select $w'_j \in \overline{W}'_j \subseteq \overline{V}$, move $w'_j$ from $\overline{W}'_j$ to $\overline{W}'_i$ and name the new partition $\overline{\mathcal{W}}''$. Then it is

$$P(\overline{W}''_i) = P(\{w'_j\}) \leq P(\{w^*_i\}) \leq P(W^*_i)$$

and

$$P(\overline{W}''_j) = P(\overline{W}'_j \backslash \{w'_j\}) \leq P(\overline{W}'_j) \leq P(W^*_j).$$

Thus, $P(\overline{W}''_i) \leq P(W_i)$ for all $i \in \{1, \ldots, k\}$. Repeat this procedure until $\overline{W}''_i \neq \emptyset$ for all $i \in \{1, \ldots, k\}$. Then $\overline{\mathcal{W}}''$ is a feasible solution for $\mathcal{I}$ and

$$s_p(\overline{\mathcal{W}}^*) > s_p(\mathcal{W}^*) \geq s_p(\overline{\mathcal{W}}''),$$

which is in contradiction to the optimality of $\overline{\mathcal{W}}^*$. □

With this result the approximation ratio of Algorithm 3.2.54 can be specified.

**Theorem 3.2.57**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinSumPP$_{\neq\emptyset}$, where $p$ is positive, monotonically decreasing and $P$ is monotonically increasing. Further, let $\mathcal{W}^*$ be an optimal solution for $\mathcal{I}$. Then Algorithm 3.2.54 finds a solution $\mathcal{W}$ for $\mathcal{I}$ with*

$$s_p(\mathcal{W}^*) \leq s_p(\mathcal{W}) \leq (1 + |V| - k)\, s_p(\mathcal{W}^*).$$

*This means Algorithm 3.2.54 is a $(1 + |V| - k)$-approximation algorithm for MinSumPP$_{\neq\emptyset}$ if $p$ is positive, monotonically decreasing and $P$ is monotonically increasing.*

*Proof*
The claim can be proven by induction on the size of $V$ in the form „$n \to n + k$“. For the base case let $|V| = k$. In this case there is just one feasible and therefore optimal partition of $V$ (each partitioning set exactly contains one element). This partition is created by Algorithm 3.2.54 as shown in Lemma 3.2.55. Thus, in this case Algorithm 3.2.54 is a 1-approximation algorithm. With

$$1 = 1 + |V| - k$$

the base case is proven.
Now let $|V| = n \geq 2k$ with $n \in \mathbb{N}$. Without loss of generality let $V = \{v_1, \dots, v_n\}$ such that $P(\{v_1\}) \geq \dots \geq P(\{v_n\})$. Thus, $v_1$ is the first and $v_n$ is the last element selected by Algorithm 3.2.54. Define $\overline{\mathcal{I}} := (\overline{V}, p, k)$ with

$$\overline{V} := V \backslash \{v_1, \dots, v_k\}.$$

Thus, $\overline{V} = \{v_{k+1}, \dots, v_n\}$ with $P(v_{k+1}) \geq \dots \geq P(v_n)$. Further,

$$P(\{\overline{v}\}) \leq P(\{v\}) \tag{3.2.8}$$

for all $\overline{v} \in \overline{V}$ and $v \in V \backslash \overline{V} = \{v_1, \dots, v_k\}$.
Let $\mathcal{W} = \{W_1, \dots, W_k\}$ be the partition of $V$ created by Algorithm 3.2.54 for $\mathcal{I}$ as input and accordingly $\overline{\mathcal{W}} = \{\overline{W}_1, \dots, \overline{W}_k\}$ the partition of $\overline{V}$ created by Algorithm 3.2.54 with $\overline{\mathcal{I}}$ as input. Since $((i-1) \bmod k)+1 = ((i+k-1) \bmod k)+1$, it is $W_i = \overline{W}_i \cup \{v_i\}$ for all $i \in \{1, \dots, k\}$. Further, let $\mathcal{W}^* = \{W_1^*, \dots, W_k^*\}$ be an optimal solution for $\mathcal{I}$ and $\overline{\mathcal{W}}^* = \{\overline{W}_1^*, \dots, \overline{W}_k^*\}$ an optimal solution for $\overline{\mathcal{I}}$.
Because of the base case

$$s_p(\overline{\mathcal{W}}) \leq \big(1 + |\overline{V}| - k\big)\, s_p(\overline{\mathcal{W}}^*)$$

holds. With Equation (3.2.8) and Lemma 3.2.56 it follows

$$s_p(\overline{\mathcal{W}}) \leq \big(1 + |\overline{V}| - k\big)\, s_p(\mathcal{W}^*).$$

Further, with Corollary 3.0.5 it is

$$\begin{aligned} s_p(\mathcal{W}) - s_p(\overline{\mathcal{W}}) &= \sum_{i=1}^{k} P(W_i) - P(\overline{W}_i) \\ &= \sum_{i=1}^{k} P(\overline{W}_i \cup \{v_i\}) - P(\overline{W}_i) \\ &\leq \sum_{i=1}^{k} P(\{v_i\}) \\ &\leq kP(\{v_1\}). \end{aligned}$$

Let without loss of generality $v_1 \in W_1^*$. Then

$$P(\{v_1\}) \leq P(W_1^*) \leq s_p(\mathcal{W}^*)$$

and thus

$$s_p(\mathcal{W}) - s_p(\overline{\mathcal{W}}) \leq kP(\{v_1\}) \leq ks_p(\mathcal{W}^*).$$

Altogether

$$\begin{aligned} s_p(\mathcal{W}) &\leq s_p(\overline{\mathcal{W}}) + ks_p(\mathcal{W}^*) \\ &\leq \left(1 + |\overline{V}| - k\right) s_p(\mathcal{W}^*) + ks_p(\mathcal{W}^*) \\ &\leq \left(1 + |\overline{V}| - k + k\right) s_p(\mathcal{W}^*) \\ &\leq (1 + |V| - k)\, s_p(\mathcal{W}^*), \end{aligned}$$

which completes the proof. □

**MinMaxPP**

For MinMaxPP a trivial approximation was already presented in Lemma 3.2.15. However, for this trivial approximation the approximation ratio still depends on the values of $p$. In this section an algorithm is developed whose approximation ratio only depends on the size of $V$ and $k$, namely $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$. The approximation algorithm also is applicable to MinMaxPP$_{\neq\emptyset}$, which is shown at the end of this section.

The idea of Algorithm 3.2.58 is to start with a set of empty partitioning sets and iteratively distribute all elements of $V$. For that a set $R$ is introduced which holds the elements which still have to be distributed. In each iteration the partitioning set with the smallest subset value is chosen and the element from $R$ which reduces the subset value the least is added.

**Remark 3.2.59**
For simplicity reasons in Algorithm 3.2.58 the minimum

$$\operatorname*{argmin}_{v \in R} P(W_1^{(l-1)} \cup \{v\})$$

**Algorithm 3.2.58** MinMaxPP approximation

**Input:** • Instance $\mathcal{I} = (V, p, k)$ of MinMaxPP

**Output:** Feasible solution $\mathcal{W} = \{W_1, \ldots, W_k\}$ for $\mathcal{I}$

```
1: $n \leftarrow |V|$
2: $R \leftarrow V$
3: for $i = 1 \rightarrow k$ do
4:     $W_i^{(0)} \leftarrow \emptyset$
5: end for
6: for $l = 1 \rightarrow n$ do
7:     $v \in \operatorname{argmin}_{v \in R} P(W_1^{(l-1)} \cup \{v\})$
8:     $W_1^{(l)} \leftarrow W_1^{(l-1)} \cup \{v\}$
9:     for $i = 2 \rightarrow k$ do
10:        $W_i^{(l)} \leftarrow W_i^{(l-1)}$
11:    end for
12:    $R \leftarrow R \backslash \{v\}$
13:    Rename $W_1^{(l)}, \ldots, W_k^{(l)}$ such that $P(W_1^{(l)}) \leq \ldots \leq P(W_k^{(l)})$
14: end for
15: return $\mathcal{W}^{(n)} = \{W_1^{(n)}, \ldots, W_k^{(n)}\}$
```

is assumed to be nonambiguous. If there is more than one element in $R$ which achieves the minimum, an indexing of $V = \{v_1, \ldots, v_n\}$ is assumed and the element with the smallest index is selected.
The same is assumed for the renaming of the partition (line 13). If there are partitioning sets with equal subset value, they are ordered like in the starting family of empty sets.

The first step is to show the correctness of Algorithm 3.2.58.

**Theorem 3.2.60**
*With an instance $\mathcal{I} = (V, p, k)$ of MinMaxPP as input Algorithm 3.2.58 runs in polynomial time in the size of the input, terminates and returns a feasible solution for $\mathcal{I}$.*

*Proof*
Since the renaming can be done in polynomial time and all involved loop constructs are `for` loops, Algorithm 3.2.58 runs in polynomial time and terminates. In each step of the `for` loop (line 6) a $v \in R$ is selected (since $R$ is finite the minimum always exists), is added to one of the sets and removed from $R$. Since

every $v \in V$ is selected exactly once, finally $\{W_1^{(n)}, \dots, W_k^{(n)}\}$ forms a partition of $V$ and thus is a feasible solution for $\mathcal{I}$. $\square$

Like in the last section the following lemma provides an upper bound for the maximum subset value if $V$ is reduced to some subset $\overline{V}$.

**Lemma 3.2.61**

*Let $\mathcal{I} = (V, p, k)$ and $\overline{\mathcal{I}} = (\overline{V}, p, k)$ be two instances of MinMaxPP, where $\overline{V} \subseteq V$ and $P$ is monotonically increasing. Further, let $\mathcal{W}^* = \{W_1^*, \dots, W_k^*\}$ be an optimal solution for $\mathcal{I}$ and $\overline{\mathcal{W}}^*$ an optimal solution for $\overline{\mathcal{I}}$. Then*

$$m_p(\overline{\mathcal{W}}^*) \leq m_p(\mathcal{W}^*).$$

*Proof*

Assumed $m_p(\overline{\mathcal{W}}^*) > m_p(\mathcal{W}^*)$. Define $\overline{\mathcal{W}}' := \{\overline{W}'_1, \dots, \overline{W}'_k\}$ with $\overline{W}'_i := W_i^* \cap \overline{V}$. Since $\mathcal{W}^*$ is a partition of $V$ and $\overline{V} \subseteq V$, $\overline{\mathcal{W}}'$ is a partition of $\overline{V}$. Further, $P(\overline{W}'_i) \leq P(W_i^*)$ because $\overline{W}'_i \subseteq W_i^*$ for all $i \in \{1, \dots, k\}$. Therefore

$$m_p(\overline{\mathcal{W}}^*) > m_p(\mathcal{W}^*) \geq m_p(\overline{\mathcal{W}}'),$$

which is in contradiction to the optimality of $\overline{\mathcal{W}}^*$. $\square$

The following lemma states that Algorithm 3.2.58 does not create empty partitioning sets if $p$ is positive.

**Lemma 3.2.62**

*With an instance $\mathcal{I} = (V, p, k)$ of MinMaxPP as input, where $|V| \geq k$ and $p$ is positive, Algorithm 3.2.58 returns a partition $\mathcal{W}^{(n)} = \{W_1^{(n)}, \dots, W_k^{(n)}\}$ of $V$ with $W_i^{(n)} \neq \emptyset$ for all $i \in \{1, \dots, k\}$.*

*Proof*

That Algorithm 3.2.58 creates a partition of $V$ already follows with Theorem 3.2.60. It is left to show that these partitioning sets are not empty.
At the beginning all partitioning sets $W_i^{(0)}$ are empty and thus $P(W_i^{(0)}) = 0$ for $i \in \{1, \dots, k\}$. In the first cycle through the `for` loop (line 6) one $v \in V$ is added to $W_1^{(0)}$. Since $p$ is positive, it is $P(W_1^{(0)} \cup \{v\}) = P(\{v\}) = p(\{v\}, v) > 0$. Thus, this set will not be selected again before all other sets contain at least one element. Since $|V| \geq k$, in the end all $k$ partitioning sets contain at least one element. $\square$

Using this result, the proof for the approximation ratio of Algorithm 3.2.58 is given in the following theorem.

**Theorem 3.2.63**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP, where $p$ is positive, monotonically decreasing and $P$ is monotonically increasing. Further, let $\mathcal{W}^*$ be an optimal solution for $\mathcal{I}$. Then Algorithm 3.2.58 finds a solution $\mathcal{W}$ for $\mathcal{I}$ with*

$$m_p(\mathcal{W}^*) \leq m_p(\mathcal{W}) \leq \left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right) m_p(\mathcal{W}^*).$$

*This means Algorithm 3.2.58 is a $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$-approximation algorithm for MinMaxPP if $p$ is positive, monotonically decreasing and $P$ is monotonically increasing.*

*Proof*
The claim can be proven by induction on the size of $V$ in the form „$n \to n+k$“. For the base case let $|V| = k$. In this case the partition, where each partitioning set exactly contains one element, is optimal. Since $P$ is monotonically increasing, the maxiumum subset value for all other partitions is larger. With Lemma 3.2.62 this partition is created by Algorithm 3.2.58. Thus, in this case, it is a 1-approximation algorithm. With

$$1 = 1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor$$

the base case is proven.
Now let $|V| = n \geq 2k$ with $n \in \mathbb{N}$. Define $\overline{\mathcal{I}} := (\overline{V}, p, k)$ with

$$\overline{V} := V \backslash \{v_{n-k+1}, \ldots, v_n\},$$

where $v_n \in V$ is the last element selected by Algorithm 3.2.58, $v_{n-1} \in V$ is the one selected before that and so on.
For $l \in \{1, \ldots, n\}$ let $\mathcal{W}^{(l)} = \{W_1^{(l)}, \ldots, W_k^{(l)}\}$ be the sets created by Algorithm 3.2.58 with $\mathcal{I}$ as input. Accordingly, let $\overline{\mathcal{W}}^{(l)} = \{\overline{W}_1^{(l)}, \ldots, \overline{W}_k^{(l)}\}$ for $l \in \{1, \ldots, n-k\}$ be the sets Algorithm 3.2.58 creates for input $\overline{\mathcal{I}}$. Due to Remark 3.2.59 it is $\overline{\mathcal{W}}^{(l)} = \mathcal{W}^{(l)}$ for $l \in \{1, \ldots, n-k\}$. Further, let $\mathcal{W}^* = \{W_1^*, \ldots, W_k^*\}$ be an optimal solution for $\mathcal{I}$ and $\overline{\mathcal{W}}^* = \{\overline{W}_1^*, \ldots, \overline{W}_k^*\}$ an opti-

mal solution for $\overline{\mathcal{I}}$. Since $|V| = n$ and $|\overline{V}| = n - k$, Algorithm 3.2.58 returns $\mathcal{W}^{(n)}$ for $\mathcal{I}$ and $\overline{\mathcal{W}}^{(n-k)}$ for $\overline{\mathcal{I}}$. Because of the base case

$$m_p(\overline{\mathcal{W}}^{(n-k)}) \leq \left(1 + \left\lfloor \frac{|\overline{V}| - 1}{k} \right\rfloor\right) m_p(\overline{\mathcal{W}}^{*}) \tag{3.2.9}$$

holds.
With Equation (3.2.9) and Lemma 3.2.61 it follows

$$m_p(\mathcal{W}^{(n-k)}) \leq \left(1 + \left\lfloor \frac{|\overline{V}| - 1}{k} \right\rfloor\right) m_p(\mathcal{W}^{*}). \tag{3.2.10}$$

With Corollary 3.0.5 it is

$$\begin{aligned} P(W_1^{(n-k)} \cup \{v_{n-k+1}\}) - P(W_k^{(n-k)}) &\leq p(\{v_{n-k+1}\}, v_{n-k+1}) \\ &\leq \max_{v \in V} p(\{v\}, v) \end{aligned}$$

and therefore

$$P(W_1^{(n-k)} \cup \{v_{n-k+1}\}) \leq P(W_k^{(n-k)}) + \max_{v \in V} p(\{v\}, v)$$

holds. Further, $\mathcal{W}^{(n-k+1)}$ still contains at least $k-1$ sets $W^{(n-k+1)} \in \mathcal{W}^{(n-k+1)}$ with $P(W^{(n-k+1)}) \leq P(W_k^{(n-k)})$, including $W_k^{(n-k)}$ itself (remember $W_k^{(n-k)}$ is the partitioning set with maximal subset value in $\mathcal{W}^{(n-k+1)}$). One of these sets is selected in the next cycle through the `while` loop. Thus, after $k-1$ steps, in which $v_{n-k+2}$, …, $v_n$ are successively added, still $P(W^{(n)}) \leq P(W_k^{(n-k)}) + \max_{v \in V} p(\{v\}, v)$ holds for all $W^{(n)} \in \mathcal{W}^{(n)}$. Therefore it is

$$m_p(\mathcal{W}^{(n)}) \leq P(W_k^{(n-k)}) + \max_{v \in V} p(\{v\}, v).$$

With Lemma 3.2.2 and Equation (3.2.10) it follows

$$\begin{aligned} m_p(\mathcal{W}^{(n)}) &\leq P(W_k^{(n-k)}) + \max_{v \in V} p(\{v\}, v) \\ &\leq m_p(\mathcal{W}^{(n-k)}) + m_p(\mathcal{W}^{*}) \\ &\leq \left(1 + \left\lfloor \frac{|\overline{V}| - 1}{k} \right\rfloor\right) m_p(\mathcal{W}^{*}) + m_p(\mathcal{W}^{*}) \\ &= \left(1 + \left\lfloor \frac{|\overline{V}| - 1}{k} \right\rfloor + 1\right) m_p(\mathcal{W}^{*}) \end{aligned}$$

$$
\begin{aligned}
&= \left(1 + \left\lfloor \frac{|\overline{V}| - 1}{k} + 1 \right\rfloor\right) m_p(\mathcal{W}^*) \\
&= \left(1 + \left\lfloor \frac{|\overline{V}| + k - 1}{k} \right\rfloor\right) m_p(\mathcal{W}^*) \\
&= \left(1 + \left\lfloor \frac{|V| - 1}{k} \right\rfloor\right) m_p(\mathcal{W}^*),
\end{aligned}
$$

which completes the proof. □

The previous result holds for instances of MinMaxPP where $p$ is positive, monotonically decreasing and $P$ is monotonically increasing. In the following it is shown that Algorithm 3.2.58 also is a $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$-approximation algorithm for MinMaxPP$_{\neq\emptyset}$ if $p$ is positive, monotonically decreasing and $P$ is monotonically increasing.
First it is shown that if an instance of MinMaxPP is interpreted as an instance of MinMaxPP$_{\neq\emptyset}$, the optimal solution value of the latter must be larger or equal. This is stated in the following lemma.

**Lemma 3.2.64**
*Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP and $\mathcal{I}' := (V, p, k)$ an instance of MinMaxPP$_{\neq\emptyset}$. Further, let $\mathcal{W}^*$ and $\mathcal{W}'^*$ optimal solutions for $\mathcal{I}$ and $\mathcal{I}'$. Then*

$$m_p(\mathcal{W}^*) \leq m_p(\mathcal{W}'^*).$$

*Proof*
Every feasible solution for $\mathcal{I}'$ is feasible for $\mathcal{I}$. □

With this knowledge the previous results can be transferred to MinMaxPP$_{\neq\emptyset}$.

**Theorem 3.2.65**
*Algorithm 3.2.58 is a $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$-approximation algorithm for MinMaxPP$_{\neq\emptyset}$ if $p$ is positive, monotonically decreasing and $P$ is monotonically increasing.*

*Proof*
Let $\mathcal{I}' = (V, p, k)$ be an instance of MinMaxPP$_{\neq\emptyset}$ and $\mathcal{W}'^*$ an optimal solution for $\mathcal{I}'$. With Lemma 3.2.62 and Theorem 3.2.63 Algorithm 3.2.58 produces a feasible solution $\mathcal{W}'$ for $\mathcal{I}'$ with

$$m_p(\mathcal{W}') \leq \left(1 + \left\lfloor \frac{|V| - 1}{k} \right\rfloor\right) m_p(\mathcal{W}^*),$$

where $\mathcal{W}^*$ is an optimal solution for the instance $\mathcal{I} = (V, p, k)$ of MinMaxPP. With Lemma 3.2.64 it is $m_p(\mathcal{W}^*) \leq m_p(\mathcal{W}'^*)$ and thus

$$m_p(\mathcal{W}') \leq \left(1 + \left\lfloor \frac{|V| - 1}{k} \right\rfloor\right) m_p(\mathcal{W}'^*),$$

which proves the claim. □

The approximation ratio of Algorithm 3.2.58 is best possible in the sense that there are instances of MinMaxPP$_{\neq\emptyset}$ for which a solution is produced that is $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$ times worse than the optimal solution. Such an instance is presented in the following paragraph. Also it has to be mentioned that even in the absence of Remark 3.2.59 the multiplicative gap of $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$ is achieved. Thus, even if there is an ambiguous choice (in line 7 or 13) and if Algorithm 3.2.58 could guess the best choice, still a solution is produced that is $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$ times worse than the optimal solution. This is also demonstrated in the following paragraph since the given example is nonambiguous.
The idea is to create an instance of MinMaxPP$_{\neq\emptyset}$, where $V$ contains two types of elements. The elements $a_i$ have a weight less but close to 1 for $i \in \{1, \ldots, k\}$. For $i \in \{1, \ldots, k-1\}$ the elements $b_i$ in general have a weight of 1 but a weight close to 0 if the corresponding $a_i$ is contained in the same partitioning set. The element $b_k$ in general has a weight of 1 but a weight smaller than all other $b_i$ if the partitioning set contains any other element. An optimal solution for this instance is the partition $\{\{a_i, b_i\}\}_{i\in\{1,\ldots,k\}}$. Algorithm 3.2.58 starts with distributing the $a_i$ elements among the partitioning sets. Afterwards $b_k$ is added to $\{a_1\}$ since it is the „cheapest“ $b_i$. In this way the optimal solution is obstructed and $b_1$ gets the high weight of 1.

**Example 3.2.66** (Worst case example for Algorithm 3.2.58)
Let $\mathcal{I} = (V, p, k)$ be an instance of MinMaxPP$_{\neq\emptyset}$ with $k \geq 2$ and the basic set $V = \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$. Further, let

$$p(W, a_i) = 1 - \frac{\varepsilon}{i}$$

for $i \in \{1, \ldots, k\}$,

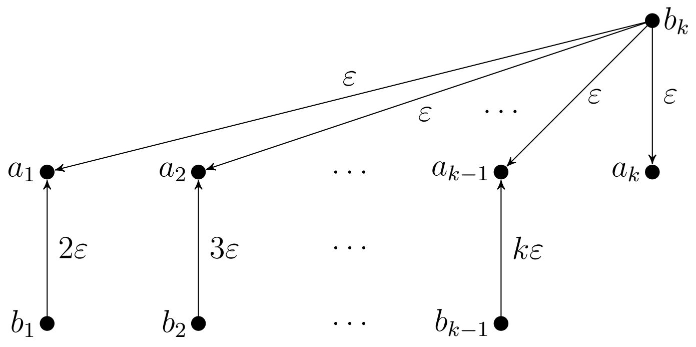

Figure 3.1: Illustration of Example 3.2.66

$$p(W, b_i) = \begin{cases} (i+1)\varepsilon & \text{if } a_i \in W \wedge |W| \geq 2 \\ 1 & \text{otherwise} \end{cases}$$

for $i \in \{1, \ldots, k-1\}$ and

$$p(W, b_k) = \begin{cases} \varepsilon & \text{if } |W| \geq 2 \\ 1 & \text{otherwise} \end{cases},$$

where $W \subseteq V$ and $\varepsilon \in \mathbb{R}^+$ such that $\varepsilon \leq \frac{1}{k+2}$.
It ist

$$1 - \frac{\varepsilon}{i} > 0$$

and thus $p$ is positive. Further, $p$ is monotonically decreasing:

Let $W' \subseteq W \subseteq V$. For $a_i \in V$ with $i \in \{1, \ldots, k\}$ it is

$$p(W', a_i) = p(W, a_i).$$

For $b_i \in V$ with $i \in \{1, \ldots, k\}$ first note that

$$|W'| \geq 2 \Rightarrow |W| \geq 2$$

and

$$a_i \in W' \Rightarrow a_i \in W$$

and therefore it either follows

$$p(W', b_i) = p(W, b_i)$$

or

$$p(W', b_i) = 1 > \frac{k}{k+2} \geq k\varepsilon \geq p(W, b_i).$$

To show that $P$ is monotonically increasing some subset $W'$ of $V$ is chosen and it is shown that adding some element $v \in V \backslash W'$ increases the subset value. Iteratively this shows the monotony of $P$:

Let $\emptyset \neq W' \subseteq V$ and $v \in V \backslash W'$. Assumed there were $w, w' \in W'$ with $w \neq w$, $p(W', w) > p(W' \cup \{v\}, w)$ and $p(W', w') > p(W' \cup \{v\}, w')$. Then of course $|W'| \geq 2$. Further, it is $w = b_i$, $w' = b_j$ with $i$, $j \in \{1, \ldots, k-1\}$ since $p(W', a_i) = p(W' \cup \{v\}, a_i)$ for $i \in \{1, \ldots, k\}$ and $p(W', b_k) = p(W' \cup \{v\}, b_k)$. Then it follows $a_i = v = a_j$ and thus $i = j$ and $w = w'$, which is a contradition. This means there is at most one $w \in W'$ with $p(W', w) > p(W' \cup \{v\}, w)$.
If there is no element $w \in W'$ with $p(W', w) > p(W' \cup \{v\}, w)$, then

$$\begin{aligned} P(W') &= \sum_{w \in W'} p(W', w) \\ &\leq \sum_{w \in W'} p(W' \cup \{v\}, w) \\ &\leq \sum_{w \in W' \cup \{v\}} p(W' \cup \{v\}, w) \\ &= P(W' \cup \{v\}) \end{aligned}$$

and nothing is to show.
Thus, assumed $w \in W'$ was the one element in $W'$ for which $p(W', w) > p(W' \cup \{v\}, w)$. Since $p(W', a_i) = p(W' \cup \{v\}, a_i)$ for $i \in \{1, \ldots, k\}$, it must be $w = b_i$ with $i \in \{1, \ldots, k\}$. This means

$$p(W', w') \leq p(W' \cup \{v\}, w')$$

for all $w' \in W' \backslash \{b_i\}$ and $\varepsilon \leq p(W' \cup \{v\}, b_i) < p(W', b_i) = 1$.

- Case $v \in \{a_1, \dots, a_k\}$:
  It is

$$\begin{aligned} & P(W' \cup \{v\}) - P(W') \\ = & \sum_{w \in W' \cup \{v\}} p(W' \cup \{v\}, w) - \sum_{w \in W'} p(W', w) \\ \geq & \, p(W' \cup \{v\}, v) + p(W' \cup \{v\}, b_i) - p(W', b_i) \\ \geq & \, (1 - \varepsilon) + \varepsilon - 1 \\ \geq & \, 0 \end{aligned}$$

  and therefore $P(W') \leq P(W' \cup \{v\})$.
- Case $v = b_j$ with $j \in \{1, \dots, k-1\}$:
  Assumed $|W'| \geq 2$. With $p(W', b_i) > p(W' \cup \{b_j\}, b_i)$ it follows $i \neq k$. This entails $a_j \notin W'$ and $a_j \in W' \cup \{b_j\}$, which is impossible. Thus, $|W'| = 1$, which means $W' = \{b_i\}$.
  With the same argument $i \neq k$ follows. Thus, $W' = \{b_k\}$. Altogether it is

$$\begin{aligned} & p(W' \cup \{v\}, v) + p(W' \cup \{v\}, b_i) - p(W', b_i) \\ = & \, p(\{b_j, b_k\}, b_j) + p(\{b_j, b_k\}, b_k) - p(\{b_k\}, b_k) \\ = & \, 1 + \varepsilon - 1 \\ > & \, 0 \end{aligned}$$

  and therefore $P(W') \leq P(W' \cup \{v\})$.
- Case $v = b_k$:
  As before it follows $|W'| = 1$ and $W' = \{b_k\}$, which is in contradiction to $b_k = v \notin W'$. Thus, this case is impossible and does not have to be considered.

Altogether, $P(W') \leq P(W' \cup \{v\})$ and thus $P$ is monotonically increasing.

An optimal solution for $\mathcal{I}$ is

$$\mathcal{W}^* := \{\{a_1, b_1\}, \{a_2, b_2\}, \dots, \{a_k, b_k\}\}:$$

It is

$$\begin{aligned}
P(\{a_i, b_i\}) &= p(\{a_i, b_i\}, a_i) + p(\{a_i, b_i\}, b_i) \\
&= \left(1 - \frac{\varepsilon}{i}\right) + (i+1)\varepsilon \\
&\leq \left(1 - \frac{\varepsilon}{(i+1)}\right) + ((i+1)+1)\varepsilon \\
&= P(\{a_{i+1}, b_{i+1}\})
\end{aligned}$$

for $i \in \{1, \dots, k-2\}$ and

$$P(\{a_k, b_k\}) = p(\{a_k, b_k\}, a_k) + p(\{a_k, b_k\}, b_k) = \left(1 - \frac{\varepsilon}{k}\right) + \varepsilon.$$

Further,

$$\begin{aligned}
& 2 \geq 1 + \frac{1}{k(k-1)} \\
\Rightarrow\ & k \geq 1 + \frac{1}{k(k-1)} \\
\Rightarrow\ & k \geq 1 + \frac{1}{k-1} - \frac{1}{k} \\
\Rightarrow\ & -\frac{1}{k-1} + (k-1) \geq -\frac{1}{k} \\
\Rightarrow\ & -\frac{\varepsilon}{k-1} + (k-1)\varepsilon \geq -\frac{\varepsilon}{k} \\
\Rightarrow\ & \left(1 - \frac{\varepsilon}{k-1}\right) + k\varepsilon \geq \left(1 - \frac{\varepsilon}{k}\right) + \varepsilon.
\end{aligned}$$

Thus,

$$P(\{a_{k-1}, b_{k-1}\}) = \left(1 - \frac{\varepsilon}{k-1}\right) + k\varepsilon \geq \left(1 - \frac{\varepsilon}{k}\right) + \varepsilon = P(\{a_k, b_k\})$$

and

$$m_p(\mathcal{W}^*) = P(\{a_{k-1}, b_{k-1}\}) = \left(1 - \frac{\varepsilon}{k-1}\right) + k\varepsilon.$$

To prove the optimality consider any other partition of $V$ and assume that for one $i \in \{1, \dots, k\}$ the elements $a_i$ and $b_i$ were in different partitioning sets. Then in the partitioning set of $b_i$ either there is an element $a_j$ with $j \in \{1, \dots, k\} \backslash \{i\}$ or there are no elements from $\{a_1, \dots a_k\}$. In the second case there must exist $i', j' \in \{1, \dots, k\}$ with $i' \neq j'$ so that $a_{i'}$ and $a_{j'}$ are in one partitioning set.
Thus, in summary there exist $i, j \in \{1, \dots, k\}$ with $i \neq j$ so that either $b_i$ and $a_j$, or $a_i$ and $a_j$ are contained in one partitioning set. It is

$$P(\{b_i, a_j\}) = 1 + \left(1 - \frac{\varepsilon}{j}\right)$$

and

$$P(\{a_i, a_j\}) = \left(1 - \frac{\varepsilon}{i}\right) + \left(1 - \frac{\varepsilon}{j}\right).$$

Thus, this partition has a maximum subset value of at least $1 + \left(1 - \frac{\varepsilon}{j}\right)$. Since

$$\begin{aligned}
&\varepsilon \leq \frac{1}{k+2} \\
&\Rightarrow (k+2)\varepsilon \leq 1 \\
&\Rightarrow k\varepsilon + 2\varepsilon - \frac{\varepsilon}{k-1} \leq 1 \\
&\Rightarrow 1 - \frac{\varepsilon}{k-1} + k\varepsilon \leq 2 - 2\varepsilon \\
&\Rightarrow 1 - \frac{\varepsilon}{k-1} + k\varepsilon \leq \left(1 - \frac{\varepsilon}{i}\right) + \left(1 - \frac{\varepsilon}{j}\right) \\
&\Rightarrow m_p(\mathcal{W}^*) \leq 1 + \left(1 - \frac{\varepsilon}{j}\right),
\end{aligned}$$

$\mathcal{W}^*$ is an optimal solution for $\mathcal{I}$.

Algorithm 3.2.58 produces the solution

$$\mathcal{W}' := \{\{a_1, b_k\}, \{a_2, b_2\}, \{a_3, b_3\}, \dots, \{a_{k-1}, b_{k-1}\}, \{a_k, b_1\}\}:$$

Algorithm 3.2.58 starts with a family of empty sets. It is

$$P(\{a_i\}) = 1 - \frac{\varepsilon}{i} < 1 - \frac{\varepsilon}{i+1} = P(\{a_{i+1}\})$$

for $i \in \{1, \dots, k-1\}$ and

$$P(\{a_k\}) = 1 - \frac{\varepsilon}{k} < 1 = P(\{b_i\})$$

for $i \in \{1, \dots, k\}$. Thus, first $a_1$ is added to one of the empty sets, then $a_2$ to one of the remaining empty sets because $P(\emptyset) = 0 < p(\{a_1\}, a_1) = P(\{a_1\})$, then $a_3$ to another empty set and so on until each set contains one of the elements of $\{a_1, \dots, a_k\}$.
After that the set $\{a_1\}$ is selected because it has the smallest subset value and $b_k$ is added since

$$P(\{a_1, b_k\}) = \left(1 + \frac{\varepsilon}{1}\right) + \varepsilon < \left(1 + \frac{\varepsilon}{1}\right) + (1+1)\varepsilon = P(\{a_1, b_1\})$$

and

$$P(\{a_1, b_k\}) = 1 + \frac{\varepsilon}{1} + \varepsilon = 1 + 2\varepsilon \leq 1 + \frac{2}{k+2} < 1 + 1 = P(\{a_1, b_i\})$$

for all $i \in \{2, \dots, k-1\}$.
The next selected set is $\{a_2\}$ since

$$P(\{a_i\}) = 1 - \frac{\varepsilon}{i} < 1 + 2\varepsilon = P(\{a_1, b_k\})$$

for $i \in \{2, \dots, k\}$, and $b_2$ is added since

$$P(\{a_2, b_2\}) = 1 - \frac{\varepsilon}{2} + (2+1)\varepsilon < 1 - \frac{\varepsilon}{2} + 1 = P(\{a_2, b_i\})$$

for all $i \in \{1, 3, 4, \dots, k-1\}$.
Iteratively for $i \in \{3, 4, \dots, k-1\}$ the set $\{a_i\}$ is selected and $b_i$ is added like

in the step before. Finally, the set $\{a_k\}$ is selected and the only remaining element is $b_1$. Therefore $\mathcal{W}'$ is the resulting partition with

$$m_p(\mathcal{W}') \geq P(\{a_k, b_1\}) = \left(1 - \frac{\varepsilon}{k}\right) + 1 = 2 - \frac{\varepsilon}{k} = \frac{2k - \varepsilon}{k}.$$

It is

$$\begin{aligned}
\frac{2k-\varepsilon}{k} &= \frac{(2k-\varepsilon)(k-1)(k-1-\varepsilon+(k-1)k\varepsilon)}{k(k-1)(k-1-\varepsilon+(k-1)k\varepsilon)} \\
&= \frac{(2k-\varepsilon)(k-1)}{k(k-1-\varepsilon+(k-1)k\varepsilon)} \cdot \frac{(k-1-\varepsilon+(k-1)k\varepsilon)}{(k-1)} \\
&= \frac{(2k-\varepsilon)(k-1)}{k(k-1-\varepsilon+(k-1)k\varepsilon)} \left(1 - \frac{\varepsilon}{k-1} + k\varepsilon\right) \\
&= \frac{(2k-\varepsilon)(k-1)}{k(k-1-\varepsilon+(k-1)k\varepsilon)} m_p(\mathcal{W}^*)
\end{aligned}$$

and therefore

$$\frac{(2k-\varepsilon)(k-1)}{k(k-1-\varepsilon+(k-1)k\varepsilon)} m_p(\mathcal{W}^*) = \frac{2k-\varepsilon}{k} \leq m_p(\mathcal{W}').$$

With

$$\begin{aligned}
\lim_{\varepsilon \to 0} \frac{(k-1)(2k-\varepsilon)}{(k-1-\varepsilon+(k-1)k\varepsilon)k} &= \frac{(k-1)2k}{(k-1)k} \\
&= 2 \\
&= 1 + \left\lfloor \frac{2k-1}{k} \right\rfloor \\
&= 1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor
\end{aligned}$$

for any $\delta > 0$ there exists an instance $\mathcal{I}'$ (choose $\varepsilon$ in $\mathcal{I}$ small enough) of MinMaxPP$_{\neq\emptyset}$ such that Algorithm 3.2.58 produces a solution for $\mathcal{I}'$ that is $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor - \delta\right)$ times larger than the optimal solution.

## 3.2.3 Numerical results

The test instances for the minimization type problems are generated like in Section 3.1.2 except that no $b$ vector is created. Instances with four different values for $|V|$ and three different values for $k$ are constructed. Like before $p$ is monotonically decreasing and $P$ is monotonically increasing and thus only the

nonempty problem versions are considered.
For MinSumPP$_{\neq\emptyset}$ as well as for MinMaxPP$_{\neq\emptyset}$ the instances are solved with the approximation algorithms (Algorithms 3.2.54 and 3.2.58) from Section 3.2.2. Further, for comparison randomized algorithms are run for the time the approximation algorithms needs. The best solution found in this time is returned. In this process only partitions with nonempty partitioning sets are considered. To get the expected solution values the randomized algorithms are run 100 times and the means are calculated.
To solve MinSumPP$_{\neq\emptyset}$ and MinMaxPP$_{\neq\emptyset}$ heuristically the Algorithms 3.2.30 and 3.2.44 from Section 3.2.1 are used, where only moves or switches are allowed which leave the partitioning sets nonempty. As start partition for the heuristics the approximations of Algorithms 3.2.54 and 3.2.58 are used. Thus, the heuristics are used to improve the results of the approximation algorithms. Of course the time to calculate the approximations is included in the runtime of the heuristics. Like previously the instances are solved by a randomized algorithm 100 times and the average of the best results is computed. Of course, this time the runtime of each run is bounded by the time the heuristic needs.
Like before, the sizes of the instances are chosen such that they can be solved in within about 60 seconds. Nevertheless, trends in runtimes and the performance of the randomized algorithms can already be detected with these instances.

All results for MinSumPP$_{\neq\emptyset}$ are summarized in Table 3.2. The runtime for the approximation algorithm is almost 0 for all instances. For the heuristic the runtime increases with increasing $|V|$ whereas size of $k$ does not have a big impact. This means only $|V|$ influences the number of possible moves or switches to reduce the target value. A further quantitative analysis of the heuristic runtime is fallacious since it strongly depends on the quality of the start partition (starting with an optimal partition for example always leads to a runtime of 0 since no moves or switches improve the solution).
In addition it is apparent that the runtime of the randomized algorithm has no impact on the quality of the average solution value. For comparison with the approximation it is allowed to run 0.001 seconds for each instance. For the heuristic it is allowed to run between 0.435 and 65.471 seconds but the change in the average target value is insignificant. Thus, in the feasible region the sections with „good“ solutions must be very small and hard to hit with random tries.

| $\|V\|$ | k | $\text{MinSumPP}_{\neq\emptyset}$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | approx. | | rand. | heur. | | rand. |
| | | time | target | target | time | target | target |
| 50 | 10 | 0.001 | 1354.4 | 1361.3 | 0.898 | 1210.4 | 1352.5 |
| | 20 | 0.001 | 1589.8 | 1559.7 | 0.703 | 1297.7 | 1545.5 |
| | 30 | 0.001 | 2004.0 | 1848.4 | 0.435 | 1718.3 | 1838.7 |
| 70 | 10 | 0.001 | 2836.0 | 2842.6 | 2.792 | 2555.7 | 2834.2 |
| | 20 | 0.001 | 3159.4 | 3174.0 | 2.840 | 2705.5 | 3168.1 |
| | 30 | 0.001 | 3609.7 | 3550.2 | 2.573 | 3094.3 | 3544.0 |
| 100 | 10 | 0.001 | 5538.5 | 5578.6 | 10.819 | 5169.6 | 5542.0 |
| | 20 | 0.001 | 6046.8 | 6124.9 | 10.443 | 5320.9 | 6074.5 |
| | 30 | 0.001 | 6495.3 | 6540.4 | 11.076 | 5463.0 | 6501.6 |
| 150 | 10 | 0.001 | 11994.2 | 12071.3 | 65.471 | 11373.5 | 11980.7 |
| | 20 | 0.001 | 12503.3 | 12604.7 | 58.241 | 11250.8 | 12524.7 |
| | 30 | 0.001 | 13347.6 | 13542.0 | 52.919 | 11814.7 | 13441.4 |

Table 3.2: $\text{MinSumPP}_{\neq\emptyset}$ results

| $\|V\|$ | k | $\text{MinSumPP}_{\neq\emptyset}$ approx. | $\text{MinSumPP}_{\neq\emptyset}$ heur. | |
|---|---|---|---|---|
| | | vs rand. | vs approx. | vs rand. |
| 50 | 10 | 1% | 11% | 11% |
| | 20 | -2% | 18% | 16% |
| | 30 | -8% | 14% | 7% |
| 70 | 10 | 0% | 10% | 10% |
| | 20 | 0% | 14% | 15% |
| | 30 | -2% | 14% | 13% |
| 100 | 10 | 1% | 7% | 7% |
| | 20 | 1% | 12% | 12% |
| | 30 | 1% | 16% | 16% |
| 150 | 10 | 1% | 5% | 5% |
| | 20 | 1% | 10% | 10% |
| | 30 | 1% | 11% | 12% |

Table 3.3: Improvement of $\text{MinSumPP}_{\neq\emptyset}$ approximation compared to randomized results and improvement of $\text{MinSumPP}_{\neq\emptyset}$ heuristic compared to approximation and randomized results

| \|V\| | k | MinMaxPP$_{\neq\emptyset}$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | approx. | | rand. | heur. | | rand. |
| | | time | target | target | time | target | target |
| 50 | 10 | 0.002 | 161.0 | 256.9 | 0.026 | 135.5 | 237.6 |
| | 20 | 0.002 | 108.0 | 177.3 | 0.031 | 80.5 | 176.0 |
| | 30 | 0.002 | 109.0 | 160.4 | 0.010 | 79.0 | 151.0 |
| 70 | 10 | 0.005 | 315.1 | 506.6 | 0.083 | 282.1 | 469.0 |
| | 20 | 0.003 | 185.8 | 323.3 | 0.108 | 157.0 | 316.5 |
| | 30 | 0.004 | 162.7 | 287.8 | 0.046 | 123.5 | 281.4 |
| 100 | 10 | 0.010 | 607.2 | 903.3 | 0.278 | 552.4 | 851.1 |
| | 20 | 0.008 | 347.8 | 649.0 | 0.223 | 299.4 | 604.1 |
| | 30 | 0.007 | 261.3 | 493.3 | 0.198 | 215.0 | 458.8 |
| 150 | 10 | 0.026 | 1277.1 | 1703.1 | 0.958 | 1196.9 | 1674.3 |
| | 20 | 0.019 | 691.4 | 1128.3 | 0.543 | 620.3 | 1100.0 |
| | 30 | 0.016 | 526.0 | 952.2 | 0.410 | 441.6 | 885.3 |

Table 3.4: MinMaxPP$_{\neq\emptyset}$ results

The results of the approximation algorithm, the heuristic and the randomized algorithms are compared in Table 3.3. In the first column the approximation results are compared to the corresponding randomized results (for $|V| = 50$ and $k = 10$, for example, the approximation result is 1% better than the average random result). It is noticeable that the approximation is not much better and in three cases even worse than with random trying. The reason for that is certainly the mediocre approximation ratio of $(1+|V|-k)$. A better approximation ratio at this point would be desirable. Nevertheless, the heuristic in average can improve the solution about 12%.

The results for MinMaxPP$_{\neq\emptyset}$ are illustrated in Table 3.4. The time to calculate the approximation does not exceed 0.026 seconds for all instances. Further, also the heuristic always runs in less than 1 second. This is a hint towards the presumption that the quality of the approximation is already quite good so that not many moves or switches can be made to improve the solution. As comparison for both approaches, the approximation and the heuristic, a randomized algorithm is tested. Like for MinSumPP$_{\neq\emptyset}$ it is conspicuous that the runtime of the randomized algorithm does not change the quality of the

| \|V\| | k | $\text{MinMaxPP}_{\neq\emptyset}$ approx. | $\text{MinMaxPP}_{\neq\emptyset}$ heur. | |
|---|---|---|---|---|
| | | vs rand. | vs approx. | vs rand. |
| 50 | 10 | 37% | 16% | 43% |
| | 20 | 39% | 25% | 54% |
| | 30 | 32% | 28% | 48% |
| 70 | 10 | 38% | 10% | 40% |
| | 20 | 43% | 15% | 50% |
| | 30 | 43% | 24% | 56% |
| 100 | 10 | 33% | 9% | 35% |
| | 20 | 46% | 14% | 50% |
| | 30 | 47% | 18% | 53% |
| 150 | 10 | 25% | 6% | 29% |
| | 20 | 39% | 10% | 44% |
| | 30 | 45% | 16% | 50% |

Table 3.5: Improvement of $\text{MinMaxPP}_{\neq\emptyset}$ approximation compared to randomized results and improvement of $\text{MinMaxPP}_{\neq\emptyset}$ heuristic compared to approximation and randomized results

solution and thus the sections with „good“ solutions in the feasible region must be very small.

The results for $\text{MinMaxPP}_{\neq\emptyset}$ are compared in Table 3.5. Already the approximation in average is 39% better than the random results whereas for $\text{MinSumPP}_{\neq\emptyset}$ the approximation value almost equals the random result. This can be explained by the much better approximation ratio of $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$. The heuristic even finds a solution that in average is 46% better than the average random result. For the approximation as well as for the heuristic with increasing $k$ the results get better compared to the results of the randomized algorithm. This makes sense for two reasons. On the one hand with increasing $k$ the number of possible partitions increases and thus it is harder for the randomized algorithm to find good solutions. On the other hand with increasing $k$ the approximation ratio $\left(1 + \left\lfloor \frac{|V|-1}{k} \right\rfloor\right)$ decreases and thus the approximation should get better. The same holds for the heuristic since it is started with the result of the approximation algorithm.

Altogether, the algorithms for $\text{MinMaxPP}_{\neq\emptyset}$ perform much better than the algorithms for $\text{MinSumPP}_{\neq\emptyset}$. It is most likely that this difference is due to the different approximation ratios. The term $\left(1+\left\lfloor\frac{|V|-1}{k}\right\rfloor\right)$ is much smaller than $(1+|V|-k)$ for large $V$ and $k$. The creation of an approximation algorithm for MinSumPP$_{\neq\emptyset}$ with improved ratio is the goal in future work.

# Chapter 4

# A partition problem with convex target function

In this chapter a special partition problem with convex target function is introduced. In contrast to the last chapter the focus is not on the complexity and approximation of the problem but on the modeling of ILPs to find optimal solutions. Further, some heuristics also based on integer linear programming are introduced. These heuristics divide the problem into smaller subproblems and combine the single solutions to get a solution for the original problem.
The partition problem in this chapter is presented as an assignment problem, in particular as a job assignment problem, in which jobs have to be assigned to machines ([3]). This makes the notation simpler. In the corresponding partition problem the set of jobs would have to be partitioned into a number of subsets, where each subset is related to a machine.
Most job scheduling problems consider linear target functions ([57], [90]). In this chapter more general piecewise linear convex target functions are assumed. In this way more practical objectives can be covered, which arise, for example, if leasing or phone contracts with included free kilometers or minutes are involved. Furthermore, in practice job scheduling problems are mostly solved with genetic or ant algorithms ([1], [83] or [15]). In this work the problem is modeled as an ILP. It is very important to find a neat formulation with as few variables and constraints as possible since otherwise runtimes may explode. Therefore, an approach is chosen where the time is not discretized. Discretizing the time would result in large programs for long periods. Also the ILP formulation is analyzed and tightened extensively so that many variables and constraints become redundant. Like this, instances with 200 jobs and 8 machine types can still be solved in less than a minute. Parts of this chapter will also be published in [9].

Since the partition problem is denoted as job assignment problem, the term „job“ has to be clarified. Also a time window version ([7]) of the assignment problem in which the starting times of the jobs are variable is introduced later. Thus, also the term „movable job“ has to be specified. This is done in the following definition.

**Definition 4.0.1**
*A tuple $j = (s, d)$ with $s \in \mathbb{N}_0$ and $d \in \mathbb{N}$ is called* job*, $s$ is called* starting time *and $d$ is called* duration *of the job.*
*A triple $k = (s^{\min}, s^{\max}, d)$ with $s^{\min}, s^{\max} \in \mathbb{N}_0$ and $d \in \mathbb{N}$, where $s^{\min} \leq s^{\max}$, is called* movable job*, $s^{\min}$ is called* lower bound for the starting time*, $s^{\max}$ is called* upper bound for the starting time *and $d$ is called* duration *of the movable job.*
*Let $K = \{(s_1^{\min}, s_1^{\max}, d_1), \ldots, (s_l^{\min}, s_l^{\max}, d_l)\}$ be a set of $l \in \mathbb{N}$ movable jobs. A set of jobs $J = \{(s_1, d_1), \ldots, (s_l, d_l)\}$ is called* feasible *for $K$ if and only if $s_i^{\min} \leq s_i \leq s_i^{\max}$ for all $i \in \{1, \ldots, l\}$. The tuple $(s_i, d_i)$ is called* corresponding *job for the movable job $(s_i^{\min}, s_i^{\max}, d_i)$ and vice versa for all $i \in \{1, \ldots, l\}$.*

**Remark 4.0.2**
For a set of $l \in \mathbb{N}$ jobs $J = \{j_1, \ldots, j_l\}$ the starting times and durations should be denoted by $s_i$ and $d_i$ for $i \in \{1, \ldots, l\}$. However, in the following for a job $j \in J$ the starting time and the duration rather are denoted by $s_j$ and $d_j$, i.e. $j = (s_j, d_j)$. In this way it is easier to address starting times and durations of arbitrary sets of jobs, which is helpful in the following definition. Thus, once $j$ denotes the real job $j \in J$ and once $j$ is just an indexing identifier.

To indicate which jobs can be processed together on one machine the concept of overlapping jobs has to be introduced. Two jobs which overlap temporally cannot be processed on the same machine since only one job can be handled at a time.

**Definition 4.0.3**
*Let $J$ be a finite set of jobs, where $s_j$ is the starting time and $d_j$ the duration of job $j \in J$. Two jobs $j, j' \in J$ are called* overlapping *if and only if there exists a number $\tau \in \mathbb{N}_0$ with*

$$s_j \leq \tau < s_j + d_j \quad \text{and} \quad s_{j'} \leq \tau < s_{j'} + d_{j'}.$$

*The variable $\tau$ is called* overlapping indicator *for $j$ and $j'$.*
*The set of jobs $J$ is called* overlapping set *if and only if there exists a number $\tau \in \mathbb{N}_0$ with*

$$s_j \leq \tau < s_j + d_j$$

*for all $j \in J$. Like before $\tau$ is called* overlapping indicator *for $J$.*
*The set of all overlapping subsets of $J$ is denoted by*

$$O_J := \{J' \subseteq J : J' \textit{ is overlapping set}\}.$$

*The size of a maximal cardinality overlapping subset of $J$ is denoted by*

$$\rho_J := \max\{|J'| : J' \in O_J\}.$$

*For a given $\tau \in \mathbb{N}_0$ a job $j \in J$ is called* active *at $\tau$ if and only if*

$$s_j \leq \tau < s_j + d_j.$$

**Remark 4.0.4**
If a set of jobs $J$ is an overlapping set, then every pair of jobs $j, j' \in J$ overlaps (the overlapping indicator of the overlapping set $\tau$ can also be used as overlapping indicator for every pair of jobs).
The maximal cardinality overlapping subset $\rho_J$ is well-defined because the empty set always is an overlapping set and $J$ is a finite set. Furthermore, $\rho_J \geq 1$ for every nonempty set of jobs $J$ since $\{j\}$ is an overlapping set for all $j \in J$.

For a subset $\overline{J}$ of a set of jobs $J$ the overlapping sets can be obtained by intersecting all overlapping sets in $J$ with $\overline{J}$. This is proven in the following lemma.

**Lemma 4.0.5**
*Let $J$ be a finite set of jobs and $\overline{J} \subseteq J$. Then*

$$O_{\overline{J}} = \bigcup_{O \in O_J} O \cap \overline{J}.$$

*Proof*
„$\subseteq$“: Let $\overline{O} \in O_{\overline{J}}$. Then $\overline{O}$ is also an overlapping set in $J$ (the same overlapping indicator as for $\overline{O}$ can be used). Thus, set $O := \overline{O} \in O_J$. It follows $\overline{O} = O \cap \overline{J} \in \bigcup_{O \in O_J} O \cap \overline{J}$.
„$\supseteq$“: Select an arbitrary $O \in O_J$. Then $O \cap \overline{J} \subseteq \overline{J}$ and $O \cap \overline{J}$ is an overlapping set in $\overline{J}$ (the same overlapping indicator as for $O$ can be used). Thus, $O \cap \overline{J} \in O_{\overline{J}}$. $\square$

The following lemma shows that as overlapping indicator for an overlapping set always the latest starting time of the jobs can be selected. This also implies that all overlapping subsets can be calculated by determining the active jobs at the starting times of the jobs.

**Lemma 4.0.6**
*If a finite set of jobs $J$ is an overlapping set, then there is a job $j' \in J$ with*

$$s_j \leq s_{j'} < s_j + d_j \quad \forall j \in J.$$

*Proof*
Let $\tau \in \mathbb{N}_0$ be an overlapping indicator for $J$. Thus,

$$s_j \leq \tau < s_j + d_j$$

for all $j \in J$. Choose $j' \in \operatorname{argmax}_{j \in J}\{s_j\}$. By definition $s_j \leq s_{j'}$ for all $j \in J$ and since $j' \in J$, also $s_{j'} \leq \tau$. Altogether $s_j \leq s_{j'} \leq \tau < s_j + d_j$ for all $j \in J$. $\square$

## 4.1 Standard formulation

The upcoming definition introduces the job assignment problem with convex target function, where the set $M$ represents the set of machines and $T$ the set of different machine types. Further, the function $\hat{t}$ specifies which machine is of which type. The matrix $r$ defines if a job can be processed by a certain type of machine or not (0 for „no“, 1 for „yes“). The goal is to find a solution or assignment $a$ which assigns each job to a machine such that no two jobs overlap. This is assured by $\rho_{a^{-1}(m)} \leq 1$ for each $m \in M$, where $a^{-1}(m)$ is the preimage of $m$ under $a$. Thus, $a^{-1}(m)$ contains all jobs that are assigned to $m$. The cost vector $c$ is composed of a fixed cost component $c_0$, which represents the costs to provide a machine and is independent of the production on this machine, and the production cost components $c_1$ and $c_2$, which represent the costs per production time. The production limit $f$ is the point where these costs increase from $c_1$ to $c_2$.

Before giving the actual definition, the structure of the cost function is introduced in the following lemma. It is verified that the target function is convex and further that it is piecewise linear. This is important for solving the problem with integer linear programming.

**Lemma 4.1.1**
*Let $f \in \mathbb{N}_0$, $c_1, c_2 \in \mathbb{R}_0^+$, with $c_1 \leq c_2$, and $x \in \mathbb{R}$. If $x \leq f$, then*

$$\max\{c_1 x, c_2 x - f(c_2 - c_1)\} = c_1 x$$

*and if $x > f$, then*

$$\max\{c_1 x, c_2 x - f(c_2 - c_1)\} = c_2 x - f(c_2 - c_1).$$

*Thus,*

$$g : \mathbb{R} \to \mathbb{R},\ x \mapsto \max\{c_1 x, c_2 x - f(c_2 - c_1)\}$$

*is a piecewise linear convex function.*

*Proof*
If $x \leq f$, then $c_2 x - f(c_2 - c_1) \leq c_2 x - x(c_2 - c_1) = c_1 x$. Otherwise if $x > f$, then $c_2 x - f(c_2 - c_1) \geq c_2 x - x(c_2 - c_1) = c_1 x$. Thus, $g$ is piecewise linear. With $c_1 f = c_2 f - f(c_2 - c_1)$ the function $g$ is continuous. Since $c_1 \leq c_2$, it also is convex. $\square$

The definition of the Convex Job Assignment Problem is given as follows.

**Definition 4.1.2** (Convex Job Assignment Problem)
*Let $J$ be a finite set of jobs, where $d_j$ is the duration of job $j \in J$. Further, let $M$ and $T$ be finite sets, $\hat{t} : M \to T$, $r \in \{0,1\}^{J \times T}$, $f \in \mathbb{N}_0$ and $c_0, c_1, c_2 \in \mathbb{R}_0^+$ with $c_1 \leq c_2$.*

*The problem to find an assignment* $a : J \to M$ *with* $\rho_{a^{-1}(m)} \leq 1$ *for all* $m \in M$ *and* $r_{j,\hat{t}(a(j))} = 1$ *for all* $j \in J$*, where*

$$c_0|a(J)| + \sum_{m\in M} \max\left\{c_1 \sum_{j\in a^{-1}(m)} d_j, c_2 \sum_{j\in a^{-1}(m)} d_j - f(c_2 - c_1)\right\}$$

*is minimal, is called* Convex Job Assignment Problem *(JAP).*

The total number of machines which process a job is the cardinality of the image of the assignment function $|a(J)|$. In the notation of a partition problem a new identifier counting the number of nonempty partitioning sets would have to be introduced. Thus, at this point it becomes evident that the assignment problem notation is superior. The number $|a(J)|$ has to be multiplied by the fixed costs $c_0$ to get the total fixed costs. The sum $\sum_{j\in a^{-1}(m)} d_j$ represents the total production time on a single machine $m$.
It has to be mentioned that the three dimensional cost vector $c$ also could depend on machines or machine types. This is often the case in practical applications. For simplicity reasons a constant vector for all machines and machine types is assumed at this point. However, further dimensions for machines or machine types can be simply added to $c$ without affecting the results of this chapter.

The following theorem provides a small insight into the complexity of the assignment problem. Even if restricted to instances with a small number of machines and machine types, JAP does not admit a polynomial-time approximation algorithm with exponential approximation ratio.

**Theorem 4.1.3**
*Unless* $\mathbb{P} = \mathbb{NP}$*, there is no polynomial-time approximation algorithm that solves JAP with approximation ratio in* $\mathcal{O}\left(2^{\mathrm{poly}(|J|,|M|,|T|,f)}\right)$ *for some polynomial* $\mathrm{poly} : \mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 1}$*. This is true even if* $|M| = 2$*,* $|T| = 2$*,* $r_{j,t} = 1$ *for all* $j \in J$ *and* $t \in T$*,* $c_0 = 0$ *and* $c_1 = 1$*.*

*Proof*
Let $\mathcal{I} = (N)$ be an instance of NPP with $N = \{n_1, \dots, n_l\}$, where $l \in \mathbb{N}$, and $\overline{N} := \sum_{i=1}^{l} n_i$ ($N$ is used for NPP instead of $S$ so that $n_l$ cannot be confused with $s_j$). Without loss of generality let every number in $N$ be even (otherwise multiply every number by 2 and solve the equivalent problem) and $\frac{\overline{N}}{2} \geq 1$. Assumed a

polynomial $Z$-approximation algorithm $\mathcal{A}$ for JAP existed with $Z \in \mathbb{R}_{\geq 1}$. Define an instance $\mathcal{J} := (J, M, T, \hat{t}, r, f, c)$ of JAP with $M := \{m_1, m_2\}$, $T := \{t_1, t_2\}$, $\hat{t}(m_1) := t_1$ and $\hat{t}(m_2) := t_2$. Thus, there are two machines $m_1$ and $m_2$ which are of type $t_1$ and $t_2$. Futher, let $f := \frac{\overline{N}}{2} \in \mathbb{N}$, $c_0 := 0$, $c_1 := 1$ and

$$c_2 := \underbrace{(Z-1)\overline{N}}_{\geq 0} + 2 > 1 = c_1.$$

Define the set $J = \{(s_j, d_j)\}_{j \in \{1,\ldots,l\}}$ of $l$ jobs with

$$(s_j, d_j) := \left( \sum_{i=1}^{i<j} n_i, n_j \right)$$

for $j \in \{1, \ldots, l\}$.
Let $(s_j, d_j)$ and $(s_{j'}, d_{j'})$ be two of these jobs with $j < j'$. Then

$$s_j + d_j = \sum_{i=1}^{i<j} n_i + n_j = \sum_{i=1}^{i<j+1} n_i \leq \sum_{i=1}^{i<j'} n_i = s_{j'}$$

and thus no $\tau \in \mathbb{N}_0$ with

$$s_j \leq \tau < s_j + d_j \quad \text{and} \quad s_{j'} \leq \tau < s_{j'} + d_{j'}$$

exists. Hence, no two jobs overlap.
Further, define $r_{j,t} := 1$ for all $j \in J$ and $t \in T$. Let $a : J \to M$ be an assignment that assigns each job to a machine. Further, let $A_i := a^{-1}(m_i)$ be the jobs assigned to machine $m_i$ with $i \in \{1, 2\}$. Thus, $A_1 \cup A_2 = J$. Since no jobs overlap, it is $\rho_{a^{-1}(m_i)} \leq 1$ for $i \in \{1, 2\}$.
If

$$\sum_{j \in A_i} d_j = \sum_{j \in A_i} n_j = \frac{\overline{N}}{2} = f$$

for $i \in \{1, 2\}$, then $\{\{n_j\}_{j \in A_1}, \{n_j\}_{j \in A_2}\}$ is a feasible partition for $\mathcal{I}$.

Further, it is

$$\begin{aligned}
&\sum_{i=1}^{2} \max\left\{c_1 \sum_{j\in A_i} d_j, c_2 \sum_{j\in A_i} d_j - f(c_2-c_1)\right\}\\
&= \sum_{i=1}^{2} \max\left\{c_1 f, c_2 f - f(c_2-c_1)\right\}\\
&= 2fc_1\\
&= 2f.
\end{aligned}$$

If without loss of generality

$$\sum_{j\in A_1} d_j = \sum_{j\in A_1} n_j > \frac{\overline{N}}{2} = f,$$

then

$$\sum_{j\in A_2} d_j = \sum_{j\in A_2} n_j = \overline{N} - \sum_{j\in A_1} n_j < \frac{\overline{N}}{2} = f.$$

Since $n_j \in \mathbb{N}$ for all $j \in J$ and $f \in \mathbb{N}$, also $\sum_{j\in A_1} n_j - f \geq 1$ holds. Thus, with Lemma 4.1.1 it is

$$\begin{aligned}
&\sum_{i=1}^{2} \max\left\{c_1 \sum_{j\in A_i} d_j, c_2 \sum_{j\in A_i} d_j - f(c_2-c_1)\right\}\\
&= c_2 \sum_{j\in A_1} n_j - f(c_2-c_1) + c_1 \sum_{j\in A_2} n_j\\
&= c_2 \sum_{j\in A_1} n_j - f(c_2-1) + \left(\overline{N} - \sum_{j\in A_1} n_j\right)\\
&= \sum_{j\in A_1} n_j(c_2-1) - f(c_2-1) + \overline{N}\\
&= \left(\sum_{j\in A_1} n_j - f\right)(c_2-1) + \overline{N}\\
&\geq (c_2-1) + \overline{N}\\
&= ((Z-1)\overline{N} + 2 - 1) + \overline{N}\\
&= Z\overline{N} + 1
\end{aligned}$$

$$\begin{aligned} &= 2Zf + 1 \\ &> 2Zf. \end{aligned}$$

Since $\mathcal{A}$ is a $Z$-approximation, a value less or equal $2Zf$ is returned by $\mathcal{A}$ for $\mathcal{J}$ if and only if $\mathcal{I}$ is solvable and a value larger than $2Zf$ if and only if $\mathcal{I}$ is infeasible. Setting $Z := 2^{\text{poly}(|M|,|T|,|J|,f)}$ with some polynomial poly : $\mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 1}$ leaves the transformation polynomial in the coding length of the input. This proves the claim. $\square$

To solve JAP an ILP is formulated in the following. The only needed decision variables are

$$x_{j,m} \in \{0,1\}$$

for all $j \in J$ and $m \in M$ and

$$w_m \in \{0,1\}$$

for all $m \in M$. The $x$ variables are in one-to-one correspondence with the assignment function $a$. Thus, a job $j$ is assigned to machine $m$ if $x_{j,m} = 1$. The $w$ variables indicate whether a machine $m$ is needed or not, i.e. whether at least one job is assigned to this machine. Thus, $w_m = 1$ if there is a $j \in J$ with $x_{j,m} = 1$.
Further, three constraints are needed. The fact that each job is assigned to exactly one machine is modeled by

$$\sum_{m \in M} x_{j,m} = 1$$

for all $j \in J$. To determine if a machine is needed the constraints

$$x_{j,m} \leq w_m$$

for all $j \in J$ and $m \in M$ are introduced. These are sufficient since in the minimization problem $w_m$ is automatically set to the smallest possible value. This is 1 if a job is assigned and to 0 if not. Also because of this the integrity of the $w$ variables can be omitted and thus $w_m \in \mathbb{R}$ for all $m \in M$.

The jobs on a machine are not allowed to overlap. This is represented by the constraints

$$\sum_{j \in O} x_{j,m} \leq 1 \tag{4.1.1}$$

for all $m \in M$ and all overlapping sets $O \in O_J$. The type restrictions are enforced by

$$x_{j,m} \leq r_{j,\hat{t}(m)}$$

for all $j \in J$ and $m \in M$.
The cost or target function assembles as

$$c_0 \sum_{m \in M} w_m + \sum_{m \in M} \max\left\{ c_1 \sum_{j \in J} d_j x_{j,m}, c_2 \sum_{j \in J} d_j x_{j,m} - f(c_2 - c_1) \right\},$$

which is not linear because of the maximum in the second summand. However, due to the piecewise linearity and the convexity (Lemma 4.1.1) and since JAP is a minimization problem this part can be linearized as follows (also see Figure 4.1). A set of new variables

$$y_m \in \mathbb{R}$$

for all $m \in M$ is introduced. With

$$c_1 \sum_{j \in J} d_j x_{j,m} \leq y_m \quad \text{and} \quad c_2 \sum_{j \in J} d_j x_{j,m} - f(c_2 - c_1) \leq y_m$$

these form an upper bound for

$$\max\left\{ c_1 \sum_{j \in J} d_j x_{j,m}, c_2 \sum_{j \in J} d_j x_{j,m} - f(c_2 - c_1) \right\}.$$

Since JAP is a minimization problem, exactly the value of the maximum is assigned to $y_m$ and the target function becomes

$$c_0 \sum_{m \in M} w_m + \sum_{m \in M} y_m.$$

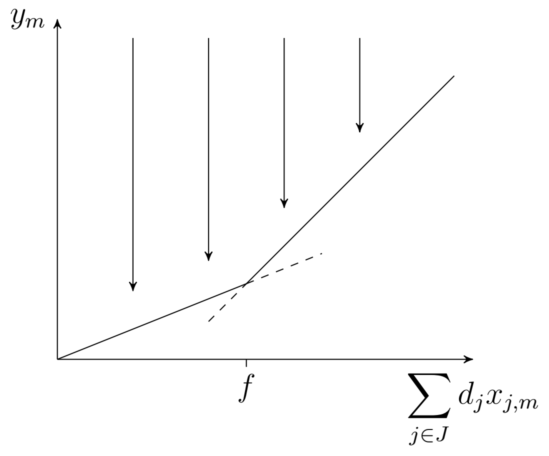

Figure 4.1: Linearizing a piecewise linear convex function

The complete ILP is presented in the following definition.

**Definition 4.1.4**
*Let $\mathcal{I} := (J, M, T, \hat{t}, r, c, f)$ be an instance of JAP, where $d_j$ is the duration of job $j \in J$. The problem JAPlin$_{\mathcal{I}}$ is defined as the following ILP:*

$$\min \; c_0 \sum_{m \in M} w_m + \sum_{m \in M} y_m$$

$$s.t.$$

$$\begin{aligned}
\sum_{m \in M} x_{j,m} &= 1 && \forall j \in J \\
x_{j,m} &\le w_m && \forall j \in J, m \in M \\
x_{j,m} &\le r_{j,\hat{t}(m)} && \forall j \in J, m \in M \\
c_1 \sum_{j \in J} d_j x_{j,m} &\le y_m && \forall m \in M \\
c_2 \sum_{j \in J} d_j x_{j,m} - f(c_2 - c_1) &\le y_m && \forall m \in M \\
\sum_{j \in O} x_{j,m} &\le 1 && \forall m \in M, O \in O_J
\end{aligned}$$

$$\begin{aligned}
x_{j,m} &\in \{0,1\} && \forall j \in J, m \in M \\
w_m &\in \mathbb{R} && \forall m \in M \\
y_m &\in \mathbb{R} && \forall m \in M
\end{aligned}$$

### 4.1.1 Movable jobs

In this section time windows for jobs are introduced. This means the starting time of a job is not fixed anymore but the job has to start within certain bounds. The problem is not just to find an assignment of jobs to machines but also to find a feasible set of starting times within the bounds. Of course, no two jobs on a machine are allowed to overlap. In the following definition and also later on the set of movable jobs is denoted by $K$.

**Definition 4.1.5** (Convex Movable Job Assignment Problem)
*Let $K$ be a finite set of movable jobs, where $d_j$ is the duration of job $j \in J$. Further, let $M$ and $T$ be finite sets, $\hat{t} : M \to T$, $r \in \{0,1\}^{K \times T}$, $c_0, c_1, c_2 \in \mathbb{R}_0^+$ with $c_1 \leq c_2$ and $f \in \mathbb{N}_0$.*
*The problem to find a feasible set of corresponding jobs $J = \{j_k\}_{k \in K}$ for $K$ and an assignment $a : J \to M$ with $\rho_{a^{-1}(m)} \leq 1$ for all $m \in M$ and $r_{k,\hat{t}(a(j_k))} = 1$ for all $k \in K$, where*

$$c_0|a(J)| + \sum_{m \in M} \max \left\{ c_1 \sum_{j \in a^{-1}(m)} d_j, c_2 \sum_{j \in a^{-1}(m)} d_j - f(c_2 - c_1) \right\}$$

*is minimal, is called* Convex Movable Job Assignment Problem *(MJAP).*

The complexity result from JAP can directly be transferred to MJAP.

**Theorem 4.1.6**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves MJAP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|K|,|M|,|T|,f)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 1}$. *This is true even if $|M| = 2$, $|T| = 2$, $r_{k,t} = 1$ for all $k \in K$ and $t \in T$, $c_0 = 0$ and $c_1 = 1$.*

*Proof*
Let $\mathcal{I} = (J, M, T, \hat{t}, r, c, f)$ be an instance of JAP. Define an instance $\mathcal{J} = (K, M, T, \hat{t}, r, c, f)$ with $K := \{(s_j, s_j, d_j)\}_{j \in J}$. Solving $\mathcal{I}$ is equivalent to solving $\mathcal{J}$ because the only feasible set of jobs for $K$ is $J$. The claim follows with Theorem 4.1.3. □

To derive an ILP formulation of MJAP the $x_{j,m}$ variables from before are substituted by $x_{k,m}$ variables which encode the assignment of job $j_k$ to a machine

$m$. Since the overlapping sets are not fixed anymore, the according restrictions are removed and a set of new variables is introduced. The variables

$$s_k \in [s_k^{\min}, s_k^{\max}] \cap \mathbb{Z}$$

encode the starting time of the job $j_k$ for each $k \in K$. The variables

$$o_{k,k'} \in \{0,1\}$$

for $k, k' \in K$ indicate if $j_k$ and $j_{k'}$ overlap and the variables

$$es_{k,k'} \in \{0,1\}$$

encode if $j_k$ ends before $j_{k'}$ begins. Two jobs $j_k$ and $j_{k'}$ do not overlap if either $j_k$ ends before $j_{k'}$ starts or $j_{k'}$ ends before $j_k$ starts.
At this point the $o$ variables are introduced to get a better understanding of the feasible region. Later it is shown that these are redundant and can be substituted by the $es$ variables. However, for now the feasible region is expressed by the following three logical expressions:

$$es_{k,k'} = 1 \Leftrightarrow s_k + d_k \leq s_{k'} \tag{4.1.2}$$

$$es_{k,k'} = 1 \vee es_{k',k} = 1 \Leftrightarrow o_{k,k'} = 0 \tag{4.1.3}$$

$$x_{k,m} = x_{k',m} = 1 \Rightarrow o_{k,k'} = 0 \tag{4.1.4}$$

Expression (4.1.4), which characterizes that jobs on the same machine are not allowed to overlap, is only needed for $k \neq k'$. Thus, all $o$ variables are only created for $k \neq k'$. Also the $es$ variables are only created for $k \neq k'$ because these are just needed to determine $o$. Since $s_k$ and $d_k$ are integer, an equivalent reformulation for Expression (4.1.2) is:

$$(es_{k,k'} = 1 \wedge s_k + d_k \leq s_{k'}) \vee (es_{k,k'} = 0 \wedge s_k + d_k \geq s_{k'} + 1) \tag{4.1.5}$$

To find a linear representation for Expression (4.1.5) the following lemma is given. Set $\mathbb{A}$ represents the feasible region for $s_k + d_k$, set $\mathbb{B}$ the feasible region for $s_{k'}$ and the set $\{0,1\}$ the region for $es_{k,k'}$. The parameter $\tau$ is chosen to be variable so that the lemma can be applied in further situations.

**Lemma 4.1.7**

*Let $\mathbb{A}, \mathbb{B} \subseteq \mathbb{R}$ be two bounded subsets of $\mathbb{R}$, $\mathbb{X} := \{0,1\} \times \mathbb{A} \times \mathbb{B}$ and $\tau \in \{0,1\}$. Further, let $H \in \mathbb{R}$ be some constant with $H \geq \max_{a\in\mathbb{A}, b\in\mathbb{B}}(a-b)$,*

$$A := \{(x,a,b) \in \mathbb{X} : x = \tau \vee a \leq b\}$$

*and*

$$B := \{(x,a,b) \in \mathbb{X} : a - b + H((1-2\tau)x + \tau - 1) \leq 0\}\,.$$

*Then $A = B$.*

*Proof*
Note that for $\tau \in \{0,1\}$ it is $\tau^2 = \tau$.
$A \subseteq B$:

Let $(x,a,b) \in A$.

- Case $x = \tau$:

$$\begin{aligned}
& a - b + H((1-2\tau)x + \tau - 1) \\
&= a - b + H((1-2\tau)\tau + \tau - 1) \\
&= a - b + H(\tau - 2\tau^2 + \tau - 1) \\
&= a - b + H(\tau - 2\tau + \tau - 1) \\
&= a - b - H \\
&\leq 0
\end{aligned}$$

- Case $x \neq \tau$: It follows $a \leq b$ and then

$$\begin{aligned}
& a - b + H((1-2\tau)x + \tau - 1) \\
&= a - b + H((1-2\tau)(1-\tau) + \tau - 1) \\
&= a - b + H(1 - \tau - 2\tau + 2\tau^2 + \tau - 1) \\
&= a - b + H(1 - \tau - 2\tau + 2\tau + \tau - 1) \\
&= a - b \\
&\leq 0.
\end{aligned}$$

Therefore $(x,a,b) \in B$.

$A \supseteq B$:

Let $(x,a,b) \in B$. If $x = \tau$, there is nothing to show. For $x \neq \tau$:

$$a = a - b + 0 + b = a - b + H((1-2\tau)x + \tau - 1) + b \leq b$$

Therefore $(x,a,b) \in A$.

□

This corollary follows directly.

**Corollary 4.1.8**
*Let $\mathbb{A}, \mathbb{B} \subseteq \mathbb{R}$ be two bounded subsets of $\mathbb{R}$ and $\mathbb{X} := \{0,1\} \times \mathbb{A} \times \mathbb{B}$. Further, let $H \in \mathbb{R}$ be some constant with $H \geq \max_{a \in \mathbb{A}, b \in \mathbb{B}} |b - a|$,*

$$A := \{(x,a,b) \in \mathbb{X} : (x = 1 \wedge a \leq b) \vee (x = 0 \wedge a \geq b + 1)\}$$

*and*

$$B := \{(x,a,b) \in \mathbb{X} : (b + 1 - a - (H+1)x \leq 0) \wedge (a - b + H(x-1) \leq 0)\}.$$

*Then $A = B$.*

*Proof*
First observe that

$$\begin{aligned}&(x = 1 \wedge a \leq b) \vee (x = 0 \wedge a \geq b + 1)\\ \Leftrightarrow\ &(x = 1 \vee a \geq b + 1) \wedge (x = 0 \vee a \leq b).\end{aligned}$$

Further, if $H \geq \max_{a \in \mathbb{A}, b \in \mathbb{B}} |b - a|$, then $(H+1) \geq \max_{a \in \mathbb{A}, b \in \mathbb{B}} |(b+1) - a| = \max_{a \in \mathbb{A}, b \in \mathbb{B}} |a - (b+1)|$. Now apply Lemma 4.1.7. □

Thus, Expression (4.1.5) can be modeled by the linear constraints

$$s_{k'} + 1 - (s_k + d_k) - (H+1)es_{k,k'} \leq 0 \tag{4.1.6}$$

and

$$(s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) \leq 0, \tag{4.1.7}$$

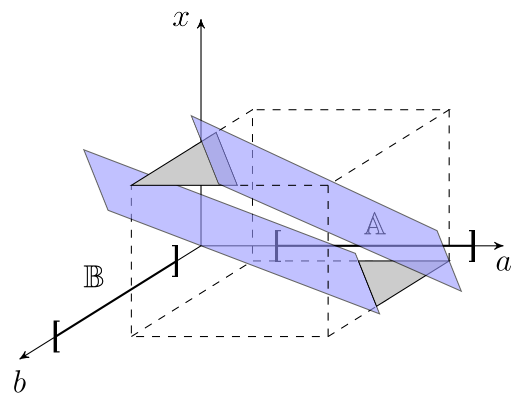

Figure 4.2: Illustration of Corollary 4.1.8

where $H \geq (s_k^{\max} + d_k) - s_{k'}^{\min}$ and $H \geq s_{k'}^{\max} - (s_k^{\min} + d_k)$. Later $H$ is chosen such that it is large enough for any pair of movable jobs $k, k' \in K$.

To get a linear representation of Expressions (4.1.3) and (4.1.4) the two following lemmas are proven. The set $\mathbb{X}$ represents the feasible region of $es_{k,k'}$, $es_{k',k}$ and $o_{k,k'}$ (for Expression (4.1.3)) or $x_{k,m}$, $x_{k',m}$ and $o_{k,k'}$ (for Expression (4.1.4)). $\tau_1$, $\tau_2$ and $\upsilon$ are constants which have to be chosen according to the relevant expression.

**Lemma 4.1.9**
*Let* $\mathbb{X} := \{0,1\}^3$ *and* $\tau_1$, $\tau_2$, $\upsilon \in \{0,1\}$. *Further, let*

$$A := \{(x_1, x_2, y) \in \mathbb{X} : (x_1 = \tau_1 \vee x_2 = \tau_2) \Rightarrow y = \upsilon\}.$$

*Then*

$$A = B := \{(x_1, x_2, y) \in \mathbb{X} : 2\upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 \\ +(2\tau_2 - 1)x_2 + 2(1 - 2\upsilon)y \leq 0\}.$$

*Proof*
For $i \in \{1,2\}$ it is $(2\tau_i - 1)x_i - \tau_i = 0$ for $x_i = \tau_i$ and $(2\tau_i - 1)x_i - \tau_i = -1$ for $x_i \neq \tau_i$.

$A \subseteq B$:

Let $(x_1, x_2, y) \in A$.

- Case $x_1 = \tau_1 \vee x_2 = \tau_2$: It follows $y = \upsilon$ and then

$$\begin{aligned}
&2\upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + 2(1 - 2\upsilon)y \\
\leq\ &2\upsilon + 2(1 - 2\upsilon)\upsilon \\
\leq\ &2\upsilon + 2\upsilon - 4\upsilon \\
=\ &0.
\end{aligned}$$

- Case $x_1 \neq \tau_1 \wedge x_2 \neq \tau_2$:

$$\begin{aligned}
&2\upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + 2(1 - 2\upsilon)y \\
=\ &2\upsilon - 2 + 2(1 - 2\upsilon)y \\
\leq\ &0
\end{aligned}$$

  For the last „$\leq$“ check all possible assignments of $\upsilon$ and $y$.

Therefore $(x_1, x_2, y) \in B$.

$A \supseteq B$:

Let $(x_1, x_2, y) \in B$.
For $x_1 \neq \tau_1 \wedge x_2 \neq \tau_2$ there is nothing to show. So let without loss of generality $x_1 = \tau_1$. Assume $y \neq \upsilon$:

$$\begin{aligned}
&2\upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + 2(1 - 2\upsilon)y \\
=\ &2\upsilon - \tau_2 + (2\tau_2 - 1)x_2 + 2(1 - 2\upsilon)(1 - \upsilon) \\
\geq\ &2\upsilon - 1 + 2(1 - 2\upsilon)(1 - \upsilon) \\
=\ &1 \\
>\ &0
\end{aligned}$$

Therefore $y = \upsilon$ and $(x_1, x_2, y) \in A$.

$\square$

**Lemma 4.1.10**
*Let $\mathbb{X} := \{0,1\}^3$ and $\tau_1$, $\tau_2$, $\upsilon \in \{0,1\}$. Further let*

$$A := \{(x_1, x_2, y) \in \mathbb{X} : (x_1 = \tau_1 \vee x_2 = \tau_2) \Leftarrow y = \upsilon\}.$$

*Then*

$$A = B := \{(x_1, x_2, y) \in \mathbb{X} : \upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 \\ +(2\tau_2 - 1)x_2 + (1 - 2\upsilon)y \geq -1\}.$$

*Proof*
For $i \in \{1, 2\}$ it is $(2\tau_i - 1)x_i - \tau_i = 0$ for $x_i = \tau_i$ and $(2\tau_i - 1)x_i - \tau_i = -1$ for $x_i \neq \tau_i$.
$A \subseteq B$

Let $(x_1, x_2, y) \in A$.

- Case $y = \upsilon$: Then it follows $x_1 = \tau_1 \vee x_2 = \tau_2$ and

$$\begin{aligned} & \upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + (1 - 2\upsilon)y \\ \geq & \upsilon - 1 + (1 - 2\upsilon)y \\ \geq & -1. \end{aligned}$$

- Case $y \neq \upsilon$:

$$\begin{aligned} & \upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + (1 - 2\upsilon)y \\ = & -\tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + 1 \\ \geq & -1 \end{aligned}$$

Therefore $(x_1, x_2, y) \in B$.

$A \supseteq B$:

Let $(x_1, x_2, y) \in B$.
For $y \neq \upsilon$ there is nothing to show. Thus, let $y = \upsilon$ and assume $x_1 \neq \tau_1 \wedge x_2 \neq \tau_2$:

$$\begin{aligned} & \upsilon - \tau_1 - \tau_2 + (2\tau_1 - 1)x_1 + (2\tau_2 - 1)x_2 + (1 - 2\upsilon)y \\ = & \upsilon + (1 - 2\upsilon)\upsilon \\ = & -2 \\ < & -1 \end{aligned}$$

Therefore $x_1 = \tau_1 \vee x_2 = \tau_2$ and $(x_1, x_2, y) \in A$.

□

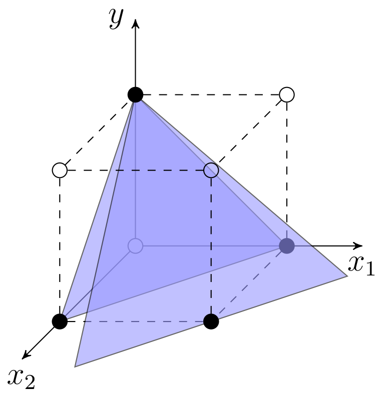

Figure 4.3: Illustration of Corollary 4.1.11

The following two corollaries are a direct result of Lemmas 4.1.9 and 4.1.10.

**Corollary 4.1.11**
*Let* $\mathbb{X} := \{0,1\}^3$. *Then*

$$\begin{aligned} &\{(x_1, x_2, y) \in \mathbb{X} : (x_1 = 1 \vee x_2 = 1) \Leftrightarrow y = 0\} \\ = \ &\{(x_1, x_2, y) \in \mathbb{X} : (x_1 + x_2 + y \geq 1) \wedge (x_1 + x_2 + 2y \leq 2)\}\,. \end{aligned}$$

*Proof*
Since

$$\begin{aligned} &(x_1 = 1 \vee x_2 = 1) \Leftrightarrow y = 0 \\ \Leftrightarrow\ &((x_1 = 1 \vee x_2 = 1) \Rightarrow y = 0) \wedge ((x_1 = 1 \vee x_2 = 1) \Leftarrow y = 0), \end{aligned}$$

Lemmas 4.1.9 and 4.1.10 can be applied with $\tau_1 = \tau_2 = 1$ and $\upsilon = 0$. □

**Corollary 4.1.12**
*Let* $\mathbb{X} := \{0,1\}^3$. *Then*

$$\begin{aligned} &\{(x_1, x_2, y) \in \mathbb{X} : x_1 = x_2 = 1 \Rightarrow y = 0\} \\ = \ &\{(x_1, x_2, y) \in \mathbb{X} : x_1 + x_2 + y \leq 2\}\,. \end{aligned}$$

*Proof*
Since $[x_1 = x_2 = 1 \Rightarrow y = 0] \Leftrightarrow [(x_1 = 0 \vee x_2 = 0) \Leftarrow y = 1]$, Lemma 4.1.10 can be applied with $\tau_1 = \tau_2 = 0$ and $\upsilon = 1$. □

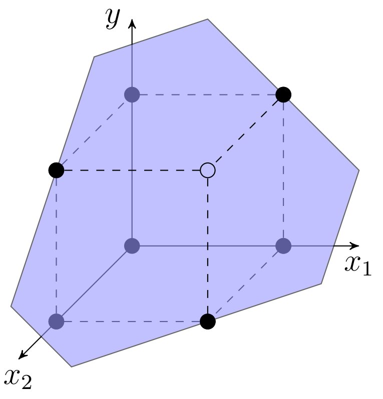

Figure 4.4: Illustration of Corollary 4.1.12

Thus, with Corollary 4.1.11 for Expression (4.1.3) and Corollary 4.1.12 for Expression (4.1.4) the following linear constraints are derived:

$$es_{k,k'} + es_{k',k} + o_{k,k'} \geq 1 \tag{4.1.8}$$
$$es_{k,k'} + es_{k',k} + 2o_{k,k'} \leq 2 \tag{4.1.9}$$
$$x_{k,m} + x_{k',m} + o_{k,k'} \leq 2 \tag{4.1.10}$$

Equations (4.1.6), (4.1.7), (4.1.8), (4.1.9) and (4.1.10) can be tightened in a way such that all $o$ variables become redundant. To see this the two following lemmas and a corollary are presented. Deliberately the notation is kept informal to carve out the key messages. The meaning of the terms and variables should be clear from their use before.

**Lemma 4.1.13**
$es_{k,k'} + es_{k',k} \leq 1$.

*Proof*
Suppose $es_{k,k'} = es_{k',k} = 1$. Then $s_k + d_k \leq s_{k'}$ and $s_{k'} + d_{k'} \leq s_k$ by Equation (4.1.7). Furthermore, $s_k < s_k + d_k$ by the definition of a job. Thus, it follows $s_{k'} + d_{k'} \leq s_k < s_k + d_k \leq s_{k'}$. This is in contradiction to the definition. Thus, $es_{k,k'} + es_{k',k} \leq 1$ holds. □

**Lemma 4.1.14**
*Equations* (4.1.8) *and* (4.1.9) *can be substituted by* $es_{k,k'} + es_{k',k} + o_{k,k'} = 1$.

*Proof*
By adding the inequality in Lemma 4.1.13 to Equation (4.1.9)

$$2es_{k,k'} + 2es_{k',k} + 2o_{k,k'} \leq 3$$

follows and therefore

$$es_{k,k'} + es_{k',k} + o_{k,k'} \leq \frac{3}{2}.$$

Since all variables are integer, this resolves to

$$es_{k,k'} + es_{k',k} + o_{k,k'} \leq 1.$$

In combination with Equation (4.1.8) this proves the claim. □

Lemma 4.1.14 shows that $es_{k,k'}$, $es_{k',k}$ and $o_{k,k'}$ are in direct dependency and thus $o_{k,k'}$ can be simply substituted.

**Corollary 4.1.15**
*Equation* (4.1.10) *is equivalent to* $x_{k,m} + x_{k',m} - (es_{k,k'} + es_{k',k}) \leq 1$

*Proof*
The equation in Lemma 4.1.14 is equivalent to $o_{k,k'} = 1 - (es_{k,k'} + es_{k',k})$. Substitution of $o_{k,k'}$ in Equation (4.1.10) proofs the claim. □

Therefore all $o$ variables can be omitted and just the following linear representation is sufficient to describe the feasible region:

$$\begin{aligned} s_{k'} + 1 - (s_k + d_k) - (H+1)es_{k,k'} &\leq 0 \\ (s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) &\leq 0 \\ x_{k,m} + x_{k',m} - (es_{k,k'} + es_{k',k}) &\leq 1 \end{aligned}$$

Further, it is easy to see that the integrality of $s$ can be omitted. For a solution in which $s$ is fractional simply use $\lfloor s \rfloor$ or $\lceil s \rceil$ as integral solution. The functions $\lfloor \cdot \rfloor$ or $\lceil \cdot \rceil$ are meant componentwise. Thus, the domain for $s_k$ is

$$s_k \in [s_k^{\min}, s_k^{\max}] \subseteq \mathbb{R}$$

as a subset of $\mathbb{R}$.

Altogether, this leads to the following definition.

**Definition 4.1.16**
*Let $\mathcal{I} := (K, M, T, \hat{t}, r, c, f)$ be an instance of MJAP, where $s_k^{\min}$ and $s_k^{\max}$ are the bounds for the starting time and $d_k$ is the duration of movable job $k \in K$. Define $H := \max_{k,k' \in K}\{(s_k^{\max} + d_k) - s_{k'}^{\min}\}$. The problem MJAPlin$_{\mathcal{I}}$ is defined as the following ILP:*

$$\min \; c_0 \sum_{m \in M} w_m + \sum_{m \in M} y_m$$

$$s.t.$$

$$\begin{aligned}
\sum_{m \in M} x_{k,m} &= 1 && \forall k \in K \\
x_{k,m} &\leq w_m && \forall k \in K, m \in M \\
x_{k,m} &\leq r_{k,\hat{t}(m)} && \forall k \in K, m \in M \\
c_1 \sum_{k \in K} d_k x_{k,m} &\leq y_m && \forall m \in M \\
c_2 \sum_{k \in K} d_k x_{k,m} - f(c_2 - c_1) &\leq y_m && \forall m \in M \\
s_{k'} + 1 - (s_k + d_k) - (H+1) es_{k,k'} &\leq 0 && \forall k \neq k' \in K \\
(s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) &\leq 0 && \forall k \neq k' \in K \\
x_{k,m} + x_{k',m} - (es_{k,k'} + es_{k',k}) &\leq 1 && \forall k \neq k' \in K, m \in M
\end{aligned}$$

$$\begin{aligned}
x_{k,m} &\in \{0,1\} && \forall k \in K, m \in M \\
w_m &\in \mathbb{R} && \forall m \in M \\
y_m &\in \mathbb{R} && \forall m \in M \\
s_k &\in [s_k^{\min}, s_k^{\max}] && \forall k \in K \\
es_{k,k'} &\in \{0,1\} && \forall k \neq k' \in K
\end{aligned}$$

## 4.1.2 Heuristic solution

Since in MJAPlin$_{\mathcal{I}}$ a quadratic amount of binary variables and constraints in the size of $|K|$ are needed, solving the ILP could take much time and a lot of memory is needed for large instances. Therefore, a heuristic which selects the starting times for the movable jobs in a separate problem is developed. Afterwards the smaller program JAPlin$_{\mathcal{I}}$ can be solved for the resulting set of jobs.

Since for each machine a fixed cost of $c_0$ has to be paid, it is reasonable to arrange the movable jobs in such a way that the number of needed machines for the resulting set of jobs $J$ is minimal. The question is how the number of needed machines can be computed. Obviously, the size of the largest overlapping set $\rho_J$ is a lower bound. In the following it is shown that $\rho_J$ machines are also sufficient to process all jobs. This is done via graph theory and thus a job graph is defined as follows.

**Definition 4.1.17**
*Let $J$ be a finite set of jobs. The graph $G_J := (V_J, E_J)$ with $V_J := \{v_j\}_{j \in J}$ and*

$$E_J := \{\{v_j, v_{j'}\} : j \neq j' \in J,\ j \text{ and } j' \text{ are overlapping}\}$$

*is called* job graph *for $J$.*

With this definition the overlapping sets in $J$ and the cliques in $G_J$ are in one-to-one correspondence. This is shown in the following lemma.

**Lemma 4.1.18**
*Let $J$ be a finite set of jobs, $J' \subseteq J$ and $G_J = (V_J, E_J)$ the job graph for $J$. The following equivalence holds:*

$$J' \text{ is an overlapping set} \Leftrightarrow \{v_j\}_{j \in J'} \text{ is a clique in } G_J$$

*Proof*
Let $s_j$ be the starting time and $d_j$ the duration of job $j \in J$.

„$\Rightarrow$“: Let $J' \subseteq J$ be an overlapping set. All pairs of jobs $j, j' \in J'$ overlap (Remark 4.0.4) and therefore

$$\{\{v_j, v_{j'}\} : v_j, v_{j'} \in J', v_j \neq v_{j'}\} \subseteq E_J.$$

„$\Leftarrow$“: Let $\{v_j\}_{j \in J'}$ be a clique in $G_J$. Then

$$\{\{v_j, v_{j'}\} : v_j, v_{j'} \in J', v_j \neq v_{j'}\} \subseteq E_J.$$

This means every two trips $j$ and $j' \in J'$ overlap. Let $\tau_{j,j'} \in \mathbb{N}_0$ be the overlapping indicator for $j, j' \in J'$,

$$\tau := \max_{j \in J'} \{s_j\}$$

and

$$u := \operatorname*{argmax}_{j \in J'} \{s_j\}.$$

By definition $\tau = s_u \leq \tau_{j,u}$ holds for all $j \in J'$ because $u$ overlaps with every other job in $J'$. Therefore

$$s_j \leq \tau \leq \tau_{j,u} < s_j + d_j$$

for all $j \in J'$ and $\tau$ is an overlapping indicator for $J'$.

□

This corollary follows directly.

**Corollary 4.1.19**
*Let $J$ be a finite set of jobs. The size of a maximum cardinality clique in $G_J$ is equal to $\rho_J$.*

*Proof*
By Lemma 4.1.18 every clique in $G_J$ corresponds to an overlapping subset of $J$ and vice versa. Thus, the size of the largest clique is equal to the size of the largest overlapping set. □

The following lemma states the fact that $G_J$ is chordal. That means every cycle of length larger than 3 has a chord (see Definition 1.3.2). Thus, if in a set $\{j_1, j_2, j_3, j_4\}$ of four jobs the jobs $j_1$ and $j_2$, $j_2$ and $j_3$, $j_3$ and $j_4$, as well as $j_4$ and $j_1$ overlap, then also either $j_1$ and $j_3$, or $j_2$ and $j_4$ must overlap.

**Lemma 4.1.20**
*Let $J$ be a finite set of jobs. Then $G_J$ is chordal.*

*Proof*
Let $s_j$ be the starting time and $d_j$ the duration of job $j \in J$. Further, let $C$ be a cycle in $G_J$ with length larger than 3. Define

$$u := \underset{j \in J}{\operatorname{argmax}} \{s_j : v_j \in C\}$$

and let $v_{u'}$ and $v_{u''}$ be the neighbors of $v_u$ in $C$ ($u' \neq u''$ since the length of the cycle is larger than 3). Let $\tau_{u,u'}$ and $\tau_{u,u''}$ be the corresponding overlapping indicators. Then $s_u$ is an overlapping indicator for $u'$ and $u''$:

$$s_{u'} \leq s_u \leq \tau_{u,u'} < s_{u'} + d_{u'} \quad \text{and} \quad s_{u''} \leq s_u \leq \tau_{u,u''} < s_{u''} + d_{u''}$$

Therefore $u'$ and $u''$ overlap and $\{v_{u'}, v_{u''}\}$ is a chord for $C$. □

This leads to the following corollary.

**Corollary 4.1.21**
*Let $J$ be a finite set of jobs. Then $G_J$ is $\rho_J$-colorable.*

*Proof*
If $G_J$ were not $\rho_J$-colorable, with Theorem 1.3.3 and Lemma 4.1.20 a $(\rho_J + 1)$-clique would exist. This is in contradiction to Corollary 4.1.19. □

Now the main result can be proven. All sets of jobs $J$ can be partitioned into $\rho_J$ subsets such that in these subsets no jobs overlap. Thus, all jobs in a subset can be assigned to one machine and therefore $\rho_J$ machines are sufficient to process all jobs.

**Theorem 4.1.22**
*Let $J$ be a set of jobs and $k := \rho_J$. Then there exists a partition $S = \{S_1, \dots, S_k\}$ of $J$ with $\rho_{S_i} \leq 1$ for all $i \in \{1, \dots, k\}$.*

*Proof*
Let $f$ be a $k$-coloring for $G_J$ (exists because of Corollary 4.1.21). Define

$$S_i := \{j \in J : f(v_j) = i\}$$

for all $i \in \{1, \dots, k\}$. Since $f$ is a coloring, there is no edge $\{v_j, v_{j'}\} \in E_J$ with $j, j' \in S_i$ for all $i \in \{1, \dots, k\}$. Hence, no two jobs overlap in $S_i$, which means $\rho_{S_i} \leq 1$ for all $i \in \{1, \dots, k\}$. □

Theorem 4.1.22 shows that the number of needed machines is exactly the size of the largest overlapping set. Thus, the starting times of the movable jobs should be chosen such that the largest overlapping set is as small as possible.
With Lemma 4.0.6 the idea is to introduce a variable for each starting time $s_{k'}$ and each movable job $k$. This variable encodes the activity of the job at the starting time. Thus,

$$p_{k,k'} \in \{0,1\}$$

for all $k, k' \in K$, where $p_{k,k'} = 1$ if the job $j_k$ is active at the starting time of $j_{k'}$. The number of active jobs at time $s_{k'}$ gives the size of the corresponding overlapping set. Thus, a variable

$$z \in \mathbb{N}$$

is introduced. With the constraints

$$\sum_{k \in K} p_{k,k'} \leq z \tag{4.1.11}$$

for all $k' \in K$ this variable forms an upper bound for the size of all overlapping sets. Thus, the minimal $z$ represents the size of the largest overlapping set. The feasible region for $p$ is expressed by the following logical expression:

$$p_{k,k'} = 1 \Leftrightarrow s_k \leq s_{k'} < s_k + d_k$$

Like before, since all variables are integer, this leads to

$$\begin{aligned}(p_{k,k'} = 1 \wedge (s_k + 1 \leq s_{k'} + 1 \leq s_k + d_k))\vee \\ (p_{k,k'} = 0 \wedge (s_k \geq s_{k'} + 1 \vee s_{k'} \geq s_k + d_k))\end{aligned} \tag{4.1.12}$$

or equivalently

$$\begin{aligned}(p_{k,k'} = 1 \vee (s_k \geq s_{k'} + 1 \vee s_{k'} \geq s_k + d_k))\wedge \\ (p_{k,k'} = 0 \vee (s_k + 1 \leq s_{k'} + 1 \leq s_k + d_k)).\end{aligned} \tag{4.1.13}$$

Unfortunately, in general these logical expressions cannot be expressed by linear inequalities without introducing further variables. Thus, in general there is

no formulation as ILP with only these variables. In the following, an extensive analysis of the feasible region is performed. Criteria are developed which categorize the cases whether such a linear representation is possible or impossible. To derive ILPs for the impossible cases more variables are introduced later. Therefore, the following results are also presented to show that these further variables are needed.
The relaxation of an ILP (integral conditions are omitted and all variables are considered continuous) has always a convex feasible region. Thus, to prove that a linear representation is impossible it is shown that Expression 4.1.12 or 4.1.13 cannot be expressed as an intersection of some convex set and the integer numbers.

In the following lemmas and corollaries the sets $\mathbb{A}$ and $\mathbb{B}$ represent the sets of feasible starting times for movable jobs $k$ and $k'$. Thus, $a_1$ and $b_1$ stand for $s_k^{\min}$ and $s_{k'}^{\min}$, and $a_2$ and $b_2$ stand for $s_k^{\max}$ and $s_{k'}^{\max}$. Further, $\nu$ or $\mu$ either represent the duration of movable job $k$ or just the number 1, as it is needed in Expressions (4.1.12) or (4.1.13).

First the part

$$s_k \geq s_{k'} + 1 \vee s_{k'} \geq s_k + d_k$$

of Expression (4.1.12) is analyzed. For that a set $A$ is defined in the following statements. This set represents the feasible region

$$F := \{(s_k, s_{k'}) \in [s_k^{\min}, s_k^{\max}] \times [s_{k'}^{\min}, s_{k'}^{\max}] : s_k \geq s_{k'} + 1 \vee s_{k'} \geq s_k + d_k\}$$

in a general form ($F$ is just an acronym in this paragraph and is not used later). In the next lemma it is assumed that $F$ only contains one point with $s_k = s_{k'} + 1$ and only one point with $s_{k'} = s_k + d_k$. As a result these points must be $(s_k^{\max}, s_{k'}^{\min})$ and $(s_k^{\min}, s_{k'}^{\max})$.

**Lemma 4.1.23**
*Let $\nu, \mu \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $\mathbb{X} := \mathbb{A} \times \mathbb{B}$ and*

$$A := \{(a, b) \in \mathbb{X} : a \geq b + \mu \vee b \geq a + \nu\}.$$

*If*

$$\{(\alpha_1, \beta_1)\} = A \cap \{(a, b) : a = b + \mu\} \ \textit{and} \ \{(\alpha_2, \beta_2)\} = A \cap \{(a, b) : b = a + \nu\}$$

*with* $\alpha_1 \neq \alpha_2$ *and* $\beta_1 \neq \beta_2$*, then*

$$(\alpha_1, \beta_1) = (a_2, b_1) \quad \textit{and} \quad (\alpha_2, \beta_2) = (a_1, b_2).$$

*Proof*
$\alpha_1 > \alpha_2$ holds:
If $\alpha_1 < \alpha_2$, then the inequalities

$$\alpha_1 < \alpha_2 \leq a_2$$

and

$$\beta_1 = \alpha_1 - \mu < \alpha_2 - \mu = \beta_2 - \nu - \mu < \beta_2 \leq b_2$$

would hold. Therefore $(\alpha_1 + 1, \beta_1 + 1) \in A \cap \{(a, b) : a = b + \mu\}$ follows, which is in contradiction to $|A \cap \{(a, b) : a = b + \mu\}\,| = 1$. So it is $a_1 \leq \alpha_2 < \alpha_1 \leq a_2$ ($\alpha_1 \neq \alpha_2$ by assumption).
$\beta_1 = b_1$ holds:
If $\beta_1 > b_1$, then the point $(\alpha_1 - 1, \beta_1 - 1)$ also would be element of $A \cap \{(a, b) : a = b + \mu\}$ which is a contradiction. $\beta_2 = b_2$ follows likewise.
$\alpha_1 = a_2$ holds:
If $\alpha_1 < a_2$, then the point $(\alpha_1 + 1, \beta_1 + 1)$ also would be element of $A \cap \{(a, b) : a = b + \mu\}$ (remember $b_1 = \beta_1 \neq \beta_2 = b_2$) which is a contradiction. $\alpha_2 = a_1$ follows likewise. □

The following lemma covers the case when there is at least one point in $F$ with $s_k > s_{k'} + 1$ and at least one point with $s_{k'} = s_k + d_k$.

**Lemma 4.1.24**
*Let* $\nu, \mu \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ *with* $a_1 \leq a_2$, $b_1 \leq b_2$ *and* $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. *Further, let* $\mathbb{X} := \mathbb{A} \times \mathbb{B}$ *and*

$$A := \{(a, b) \in \mathbb{X} : a \geq b + \mu \vee b \geq a + \nu\}.$$

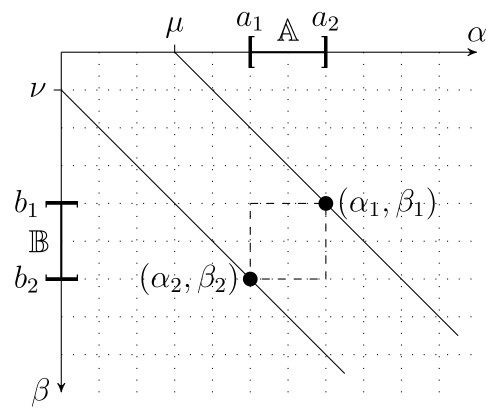

Figure 4.5: Illustration of Lemma 4.1.23

*If there exist*

$$(\alpha_1, \beta_1) \in A \cap \{(a, b) : a > b + \mu\} \quad \textit{and} \quad (\alpha_2, \beta_2) \in A \cap \{(a, b) : b \geq a + \nu\}$$

*(i) with $\alpha_1 = \alpha_2$ or $\beta_1 = \beta_2$, then*

$$|A \cap \{(a, b) : a = b + \mu\}| \geq 1.$$

*(ii) with $\alpha_1 \neq \alpha_2$ and $\beta_1 \neq \beta_2$, then*

$$|A \cap \{(a, b) : a = b + \mu\}| \geq 2.$$

*Proof*

(i) Just the case for $\alpha_1 = \alpha_2$ is demonstrated:
Define $(\alpha_1', \beta_1') := (\alpha_1, \alpha_1 - \mu)$. Then

$$b_1 \leq \beta_1 + \mu - \mu < \underbrace{\alpha_1 - \mu}_{=\beta_1'} = \alpha_2 - \mu \leq \beta_2 - \nu - \mu < b_2$$

and thus $(\alpha_1', \beta_1') \in A \cap \{(a, b) : a = b + \mu\}$.

(ii) Two points lying in $A \cap \{(a, b) : a = b + \mu\}$ are constructed regarding three cases:

- Case $\beta_2 < \beta_1$: Define $(\alpha_1', \beta_1') := (\beta_1 - 1 + \mu, \beta_1 - 1)$ and $(\alpha_1'', \beta_1'') := (\beta_1 + \mu, \beta_1)$. With

$$a_1 < \alpha_2 + \nu + \mu \leq \beta_2 + \mu < \underbrace{\beta_1 + \mu}_{=\alpha_1''} < \alpha_1 \leq a_2$$

and

$$b_1 \leq \beta_2 < \underbrace{\beta_1}_{=\beta_1''} \leq b_2$$

it follows $(\alpha_1'', \beta_1'') \in A \cap \{(a,b) : a = b + \mu\}$. In the same way it can be shown that $(\alpha_1', \beta_1') \in A \cap \{(a,b) : a = b + \mu\}$.

- Case $\beta_1 < \beta_2$ and $\alpha_2 < \alpha_1$: Define $\alpha_1' := \max\{\alpha_2, \beta_1 + \mu\}$, $\beta_1' := \alpha_1' - \mu$ and $(\alpha_1'', \beta_1'') := (\alpha_1' + 1, \beta_1' + 1)$. With

$$a_1 \leq \alpha_2 \leq \alpha_1' \leq \begin{Bmatrix} \alpha_2 \\ \beta_1 + \mu \end{Bmatrix} < \alpha_1 \leq a_2$$

and

$$\begin{aligned} b_1 \leq \beta_1 + \mu - \mu \leq \underbrace{\alpha_1' - \mu}_{=\beta_1'} \leq & \begin{Bmatrix} \alpha_2 - \mu \\ \beta_1 + \mu - \mu \end{Bmatrix} \\ \leq & \begin{Bmatrix} \beta_2 - \nu - \mu \\ \beta_1 \end{Bmatrix} < \beta_2 \leq b_2 \end{aligned}$$

it follows $(\alpha_1', \beta_1') \in A \cap \{(a,b) : a = b + \mu\}$. In the same way it can be shown that $(\alpha_1'', \beta_1'') \in A \cap \{(a,b) : a = b + \mu\}$.

- Case $\beta_1 < \beta_2$ and $\alpha_1 < \alpha_2$: Define $(\alpha_1', \beta_1') := (\alpha_1, \alpha_1 - \mu)$ and $(\alpha_1'', \beta_1'') := (\alpha_1' + 1, \beta_1' + 1)$. With

$$a_1 \leq \underbrace{\alpha_1}_{=\alpha_1'} < \alpha_2 \leq a_2$$

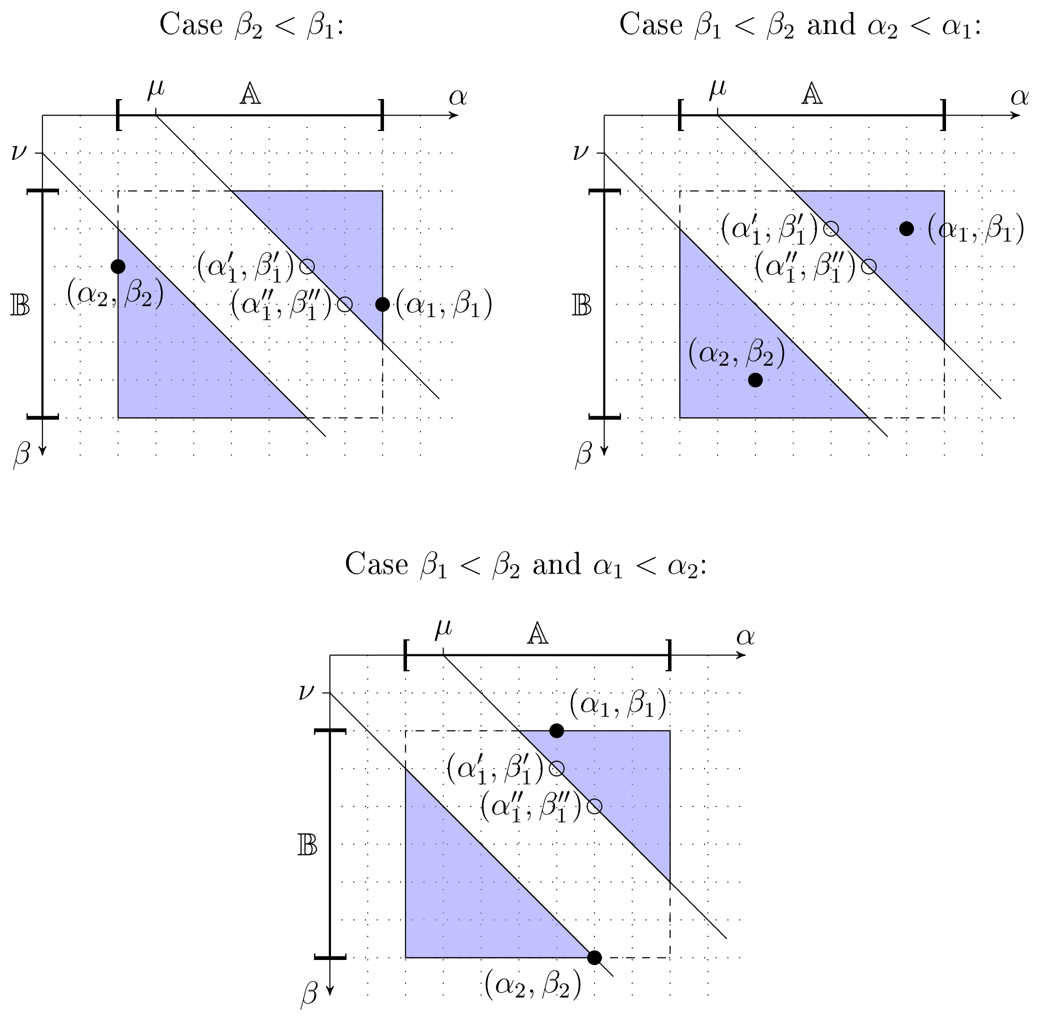
Case $\beta_2 < \beta_1$:
Case $\beta_1 < \beta_2$ and $\alpha_2 < \alpha_1$:
Case $\beta_1 < \beta_2$ and $\alpha_1 < \alpha_2$:
$\mu$
$\mathbb{A}$
$\alpha$
$\nu$
$\mathbb{B}$
$\beta$
$(\alpha_1, \beta_1)$
$(\alpha_2, \beta_2)$
$(\alpha_1', \beta_1')$
$(\alpha_1'', \beta_1'')$

Figure 4.6: Illustration of Lemma 4.1.24 (ii)

and

$$b_1 \leq \beta_1 < \underbrace{\alpha_1 - \mu}_{=\beta_1'} < \alpha_2 - \mu \leq \beta_2 - \nu - \mu < b_2$$

it follows $(\alpha_1', \beta_1') \in A \cap \{(a, b) : a = b + \mu\}$. In the same way it can be shown that $(\alpha_1'', \beta_1'') \in A \cap \{(a, b) : a = b + \mu\}$.

□

Using this result and Lemma 4.1.23 the next corollary follows directly. If $F$ contains only one point with $s_k = s_{k'} + 1$ and only one point with $s_{k'} = s_k + d_k$, then $F$ contains no other points.

**Corollary 4.1.25**

*Let $\nu, \mu \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $\mathbb{X} := \mathbb{A} \times \mathbb{B}$ and*

$$A := \{(a,b) \in \mathbb{X} : a \geq b + \mu \vee b \geq a + \nu\}.$$

*If*

$$\{(\alpha_1, \beta_1)\} = A \cap \{(a,b) : a = b + \mu\} \text{ and } \{(\alpha_2, \beta_2)\} = A \cap \{(a,b) : b = a + \nu\}$$

*with $\alpha_1 \neq \alpha_2$ and $\beta_1 \neq \beta_2$, then*

$$\{(a_2, b_1)\} = A \cap \{(a,b) : a \geq b + \mu\} \text{ and } \{(a_1, b_2)\} \ = A \cap \{(a,b) : b \geq a + \nu\}.$$

*Proof*

If there existed a point $(\alpha, \beta) \in A \cap \{(a,b) : a > b + \mu\}$, with Lemma 4.1.24 $|A \cap \{(a,b) : a = b + \mu\}| \geq 2$ would follow. This is a contradiction. Thus, $\{(\alpha_1, \beta_1)\} = A \cap \{(a,b) : a \geq b + \mu\}$. With Lemma 4.1.23 it follows $(\alpha_1, \beta_1) = (a_2, b_1)$.
The equality $\{(a_2, b_2)\} = A \cap \{(a,b) : b \geq a + \nu\}$ can be shown in the same way. □

To shorten the proof of the upcoming Lemma 4.1.27 the following lemma is given first.

**Lemma 4.1.26**

*Let $P_1 = (\alpha_1, \beta_1)$, $P_2 = (\alpha_2, \beta_2)$, $P_3 = (\alpha_3, \beta_3)$ and $P = (\alpha, \beta) \in \mathbb{Z}^2$ be four points with $P_2 = P_1 + (1,1)$ and $P = P_1 + (0,1)$. If $\alpha_3 \leq \alpha$ and $\beta_3 \geq \beta$, then $P$ lies in the convex hull of $P_1$, $P_2$ and $P_3$.*

*Proof*

As proof a simple picture is given in Figure 4.7. □

The following lemma describes the case where $F$ contains at least three points. One of these points satisfies $s_k \geq s_{k'} + 1$ and another $s_{k'} \geq s_k + d_k$. This lemma states the first negative result since in this case $F$ cannot be expressed as intersection of a convex set and the integer numbers.

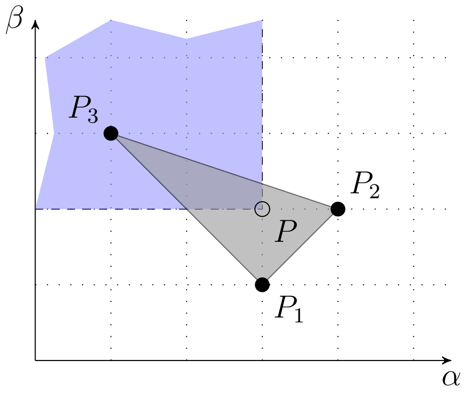

Figure 4.7: Proof of Lemma 4.1.26

**Lemma 4.1.27**
*Let $\mu, \nu \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $\mathbb{X} := \mathbb{A} \times \mathbb{B}$ and*

$$A := \{(a, b) \in \mathbb{X} : a \geq b + \mu \vee b \geq a + \nu\}.$$

*If there exist three points $(\alpha_1, \beta_1)$, $(\alpha_2, \beta_2)$ and $(\alpha_3, \beta_3) \in A$, where $\alpha_1 \geq \beta_1 + \mu$ and $\beta_3 \geq \alpha_3 + \nu$, then there is no convex set $C \subseteq \mathbb{R}^2$ with $A = C \cap \mathbb{Z}^2$.*

*Proof*
The goal is to find a point $(\alpha, \beta) \in \mathbb{Z}^2$ in the convex hull of points in $A$ but $(\alpha, \beta) \notin A$. Thus, every convex set $C$ with $A \subseteq C$ also would contain $(\alpha, \beta)$ and therefore $A \neq C \cap \mathbb{Z}^2$.
Without loss of generality $\alpha_2 \geq \beta_2 + \mu$ can be assumed due to the symmetry of $A$. With Lemma 4.1.24 the following expressions are well-defined:

$$\begin{aligned}
(\alpha_2', \beta_2') &:= \operatorname*{argmax}_{(a,b) \in \mathbb{X}} \{a : a = b + \mu\} \in A, \\
(\alpha_1', \beta_1') &:= (\alpha_2' - 1, \beta_2' - 1) \in A, \\
(\alpha_3', \beta_3') &:= \operatorname*{argmax}_{(a,b) \in \mathbb{X}} \{a : b = a + \nu\} \in A
\end{aligned}$$

$\alpha_3' \leq \alpha_2'$ and $\beta_2' \leq \beta_3'$ follow immediately:
With $\alpha_3' > \alpha_2'$ the inequality

$$b_1 \leq \beta_2' = \alpha_2' - \mu < \alpha_3' - \mu = \beta_3' - \nu - \mu < \beta_3' \leq b_2$$

would follow and thus $(\alpha_3', \alpha_3' - \mu) \in \mathbb{X}$. This is in contradiction to the definition of $(\alpha_2', \beta_2')$.
Since

$$\underset{(a,b)\in\mathbb{X}}{\operatorname{argmax}} \{a : a = b + \mu\} = \underset{(a,b)\in\mathbb{X}}{\operatorname{argmax}} \{b : a = b + \mu\},$$

in the same way $\beta_2' \leq \beta_3'$ can be shown.

The point $(\alpha, \beta)$ is created regarding the following two cases:

- Case $\alpha_2' = \alpha_3'$: Define $(\alpha, \beta) := (\alpha_2', \beta_2' + 1)$. Therefore

  $$\alpha_2' = \alpha = \alpha_3' \quad \text{and} \quad \beta_2' < \underbrace{\beta_2' + 1}_{=\beta} < \beta_2' + \mu + \nu = \alpha_2' + \nu = \alpha_3' + \nu = \beta_3'.$$

  Thus, $(\alpha, \beta)$ is a convex combination of $(\alpha_2', \beta_2')$ and $(\alpha_3', \beta_3')$. Furthermore

  $$\alpha = \alpha_2' = \beta_2' + \mu < \beta_2' + 1 + \mu = \beta + \mu$$

  and

  $$\beta = \beta_2' + 1 < \beta_2' + \mu + \nu = \alpha_2' + \nu = \alpha + \nu.$$

  Thus, $(\alpha, \beta) \notin A$ as required.

- Case $\alpha_2' > \alpha_3'$: It follows $\alpha_1' = \alpha_2' - 1 \geq \alpha_3'$ and remember $\beta_2' \leq \beta_3'$. Define $(\alpha, \beta) := (\alpha_2' - 1, \beta_2') = (\alpha_1', \beta_1' + 1)$. Then $\alpha_3' \leq \alpha$ and $\beta_3' \geq \beta$ and thus with Lemma 4.1.26 the point $(\alpha, \beta)$ is a convex combination of $(\alpha_1', \beta_1')$, $(\alpha_2', \beta_2')$ and $(\alpha_3', \beta_3')$. Further,

  $$\alpha = \alpha_2' - 1 = \beta_2' + \mu - 1 < \beta_2' + \mu = \beta + \mu$$

  and

  $$\beta = \beta_2' < \beta_2' + \mu - 1 + \nu = \alpha_2' - 1 + \nu = \alpha + \nu.$$

  Thus, $(\alpha, \beta) \notin A$ as required.

□

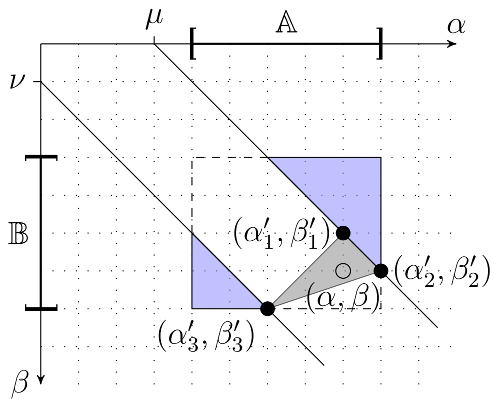

Figure 4.8: Illustration of Lemma 4.1.27

To apply this result to Expression (4.1.12) one dimension for $p_{k,k'}$ has to be added. This is done in the following theorem.

**Theorem 4.1.28**
*Let $\delta \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $\mathbb{X} := \{0, 1\} \times \mathbb{A} \times \mathbb{B}$ and*

$$\begin{aligned} A := \{(x, a, b) \in \mathbb{X} : \, & (x = 0 \wedge (a \geq b + 1 \vee b \geq a + \delta)) \vee \\ & (x = 1 \wedge (a \leq b \wedge b + 1 \leq a + \delta))\}. \end{aligned}$$

*If there exist three points $(0, \alpha_1, \beta_1)$, $(0, \alpha_2, \beta_2)$ and $(0, \alpha_3, \beta_3) \in A$, where $\alpha_1 \geq \beta_1 + 1$ and $\beta_3 \geq \alpha_3 + \delta$, then there is no convex set $C \subseteq \mathbb{R}^3$ with $A = C \cap \mathbb{Z}^3$.*

*Proof*
Using the same notation as in the proof of Lemma 4.1.27 (with $\mu = 1$ and $\nu = \delta$) a point $(0, \alpha, \beta)$ in the convex hull of $(0, \alpha'_1, \beta'_1)$, $(0, \alpha'_2, \beta'_2)$ and $(0, \alpha'_3, \beta'_3) \in A$ with $(0, \alpha, \beta) \notin A$ can be created. Therefore, there is no convex set $C \subseteq \mathbb{R}^3$ with $A = C \cap \mathbb{Z}^3$. $\square$

Thus, if $F$ contains more than two points of which one satisfies $s_k \geq s_{k'} + 1$ and another $s_{k'} \geq s_k + d_k$, modeling Expression (4.1.12) in an ILP without introducing further variables is impossible.

In the following the case where $F$ contains only two points is studied. The next helping lemma characterizes in what case the convex hull of two points in $\mathbb{Z}^2$ contains a point that lies in $\mathbb{Z}^2$ again.

**Lemma 4.1.29**
*Let* $(\alpha_1, \beta_1)$, $(\alpha_2, \beta_2) \in \mathbb{Z}^2$ *with* $\alpha_1 \neq \alpha_2$ *and*

$$\hat{r} := \min\left\{ r \in \mathbb{N} : r\frac{\beta_2 - \beta_1}{\alpha_2 - \alpha_1} \in \mathbb{Z} \right\}.$$

*Then there exists* $\lambda \in (0,1)$ *with* $\lambda(\alpha_2, \beta_2) + (1-\lambda)(\alpha_1, \beta_1) \in \mathbb{Z}^2$ *if and only if* $\hat{r} < |\alpha_2 - \alpha_1|$.

*Proof*
First note $\underbrace{|\alpha_2 - \alpha_1|}_{\in \mathbb{N}} \frac{\beta_2 - \beta_1}{\alpha_2 - \alpha_1} \in \mathbb{Z}$. Thus, $\hat{r}$ is well-defined.

„⇒“: Let $\lambda \in (0,1)$ with $\lambda(\alpha_2, \beta_2) + (1-\lambda)(\alpha_1, \beta_1) \in \mathbb{Z}^2$. Define $r := \lambda|\alpha_2 - \alpha_1|$. Then

$$\begin{aligned}
&\frac{r}{|\alpha_2 - \alpha_1|}\alpha_2 + \left(1 - \frac{r}{|\alpha_2 - \alpha_1|}\right)\alpha_1 \in \mathbb{Z} \\
\Rightarrow\ &\alpha_1 + r\frac{\alpha_2 - \alpha_1}{|\alpha_2 - \alpha_1|} \in \mathbb{Z} \\
\Rightarrow\ &\alpha_1 + r \operatorname{sgn}(\alpha_2 - \alpha_1) \in \mathbb{Z} \\
\Rightarrow\ &r \in \mathbb{Z}
\end{aligned}$$

and since $0 < \lambda|\alpha_2 - \alpha_1| = r$, it is $r \in \mathbb{N}$. Further,

$$\begin{aligned}
&\frac{r}{|\alpha_2 - \alpha_1|}\beta_2 + \left(1 - \frac{r}{|\alpha_2 - \alpha_1|}\right)\beta_1 \in \mathbb{Z} \\
\Rightarrow\ &\beta_1 + r\frac{\beta_2 - \beta_1}{|\alpha_2 - \alpha_1|} \in \mathbb{Z} \\
\Rightarrow\ &r\frac{\beta_2 - \beta_1}{|\alpha_2 - \alpha_1|} \in \mathbb{Z}.
\end{aligned}$$

Therefore, $\hat{r} \leq r = \lambda|\alpha_2 - \alpha_1| < |\alpha_2 - \alpha_1|$.

„⇐“: Define

$$\lambda := \frac{\hat{r}}{|\alpha_2 - \alpha_1|} \in (0,1).$$

Then like in ⇒“ it follows $\lambda(\alpha_2, \beta_2) + (1-\lambda)(\alpha_1, \beta_1) \in \mathbb{Z}^2$.

□

This result can be also used as a computation instruction to find an integer point in the convex hull of two integer points. As coefficient for the convex combination use $\lambda = \hat{r}/|\alpha_2 - \alpha_1|$. With this lemma the following theorem can be proven. If $F$ only contains two points with $s_k = s_{k'} + 1$ and $s_{k'} = s_k + d_k$ and no integer points lie in the convex hull of these points, then Expression (4.1.12) can be modeled by linear inequalities. These inequalities are also provided in the theorem. Again, otherwise formulating Expression (4.1.12) in an ILP is impossible without introducing further variables.

**Theorem 4.1.30**
*Let $\delta \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $\mathbb{X} := \{0, 1\} \times \mathbb{A} \times \mathbb{B}$ and*

$$A := \{(x, a, b) \in \mathbb{X} : (x = 0 \wedge (a \geq b + 1 \vee b \geq a + \delta)) \vee \\ (x = 1 \wedge (a \leq b \wedge b + 1 \leq a + \delta))\}.$$

*(i) If*

$$\{(0, \alpha_1, \beta_1)\} = A \cap \{(0, a, b) : a = b + 1\}$$

*and*

$$\{(0, \alpha_2, \beta_2)\} = A \cap \{(0, a, b) : b = a + \delta\}$$

*with $\alpha_1 \neq \alpha_2$ and*

$$\hat{r} := \min\left\{ r \in \mathbb{N} : r\frac{\beta_2 - \beta_1}{\alpha_2 - \alpha_1} \in \mathbb{Z} \right\} \geq |\alpha_2 - \alpha_1|,$$

*then $A = B$, where*

$$\begin{aligned} B := \{(x, a, b) \in \mathbb{X} : \\ &(x - a + b \leq \delta) \wedge \\ &(x + a - b \leq 1) \wedge \\ &((a_2 - a_1)(b_2 - b_1)x + (b_2 - b_1)a + (a_2 - a_1)b \geq a_2 b_2 - a_1 b_1) \wedge \\ &((a_2 - a_1)(b_2 - b_1)x + (b_1 - b_2)a + (a_1 - a_2)b \geq a_1 b_1 - a_2 b_2)\}. \end{aligned}$$

*(ii) If there exist*

$$(0,\alpha_1,\beta_1) \in A \cap \{(0,a,b) : a = b+1\}$$

*and*

$$(0,\alpha_2,\beta_2) \in A \cap \{(0,a,b) : b = a+\delta\}$$

*with either* $\alpha_1 = \alpha_2$ *or*

$$\hat{r} := \min\left\{ r \in \mathbb{N} : r\frac{\beta_2-\beta_1}{\alpha_2-\alpha_1} \in \mathbb{Z} \right\} < |\alpha_2 - \alpha_1|,$$

*then there is no convex set* $C \subseteq \mathbb{R}^3$ *with* $A = C \cap \mathbb{Z}^3$.

*Proof*

The parts (i) and (ii) are proven separately.

(i) First $\beta_1 \neq \beta_2$ is shown:

With $\beta_1 = \beta_2$ the inequality

$$\alpha_1 = \beta_1 + \mu = \beta_2 + \mu = \alpha_2 + \nu + \mu \geq \alpha_2 + 2$$

and $\hat{r} = 1$ would hold. Therefore, $1 = \hat{r} \geq |\alpha_2 - \alpha_1| \geq 2$, which is a contradiction. Thus, $\beta_1 \neq \beta_2$.

$A \subseteq B$:

Let $(x,a,b) \in A$. Two cases have to be considered. Either $x = 0$ or $x = 1$:

- Case $x = 0$: With Lemma 4.1.23 and Corollary 4.1.25 two cases follow:
  - Case $(x,a,b) = (0,\alpha_1,\beta_1) = (0,a_2,b_1)$: Then it is $a_2 = b_1 + 1$.

$$\begin{aligned} x - a + b &= 0 - a_2 + b_1 = -(b_1+1) + b_1 = -1 \leq \delta \\ x + a - b &= 0 + a_2 - b_1 = (b_1+1) - b_1 = 1 \\ (a_2-a_1)&(b_2-b_1)x + (b_2-b_1)a + (a_2-a_1)b \\ &= (b_2-b_1)a_2 + (a_2-a_1)b_1 \\ &= a_2b_2 - a_1b_1 \end{aligned} \tag{4.1.14}$$

$$\begin{aligned}
&(a_2 - a_1)(b_2 - b_1)x + (b_1 - b_2)a + (a_1 - a_2)b \\
&\qquad = (b_1 - b_2)a_2 + (a_1 - a_2)b_1 \\
&\qquad = a_1b_1 - a_2b_2
\end{aligned}$$

- Case $(x, a, b) = (0, \alpha_2, \beta_2) = (0, a_1, b_2)$: Then it is $b_2 = a_1 + \delta$.

$$\begin{aligned}
&x - a + b = 0 - a_1 + b_2 = -b_2 + \delta + b_2 = \delta \\
&x + a - b = 0 + a_1 - b_2 = b_2 - \delta - b_2 = -\delta \le 1 \\
&(a_2 - a_1)(b_2 - b_1)x + (b_2 - b_1)a + (a_2 - a_1)b \\
&\qquad = (b_2 - b_1)a_1 + (a_2 - a_1)b_2 \\
&\qquad = a_2b_2 - a_1b_1 \\
&(a_2 - a_1)(b_2 - b_1)x + (b_1 - b_2)a + (a_1 - a_2)b \\
&\qquad = (b_1 - b_2)a_1 + (a_1 - a_2)b_2 \\
&\qquad = a_1b_1 - a_2b_2
\end{aligned} \tag{4.1.15}$$

- Case $x = 1$: It follows $a \le b$ and $b + 1 \le a + \delta$.

$$\begin{aligned}
&x - a + b = b + 1 - a \le a + \delta - a = \delta \\
&x + a - b = 1 + a - b \le 1 + b - b = 1 \\
&(a_2 - a_1)(b_2 - b_1)x + (b_2 - b_1)a + (a_2 - a_1)b \\
&\qquad \ge (a_2 - a_1)(b_2 - b_1) + (b_2 - b_1)a_1 + (a_2 - a_1)b_1 \\
&\qquad = a_2b_2 - a_1b_1 \\
&(a_2 - a_1)(b_2 - b_1)x + (b_1 - b_2)a + (a_1 - a_2)b \\
&\qquad \ge (a_2 - a_1)(b_2 - b_1) + (b_1 - b_2)a_2 + (a_1 - a_2)b_2 \\
&\qquad = a_1b_1 - a_2b_2
\end{aligned}$$

Therefore, $(x, a, b) \in B$.

$B \subseteq A$:

Let $(x, a, b) \in B$.

- Case $x = 0$:
  Assumed $a < b + 1$ and $b < a + \delta$. From the two inequalities

$$(b_2 - b_1)a + (a_2 - a_1)b \ge a_2b_2 - a_1b_1$$

and

$$(b_1 - b_2)a + (a_1 - a_2)b \geq a_2 b_2 - a_1 b_1$$

it follows

$$(b_2 - b_1)a + (a_2 - a_1)b = a_2 b_2 - a_1 b_1.$$

With Equations (4.1.14) and (4.1.15) this shows that the points $(0, a_2, b_1)$, $(0, a_1, b_2)$ and $(0, a, b)$ lie on a line. Therefore, it exists $\lambda \in \mathbb{R}$ with

$$(0, a, b) = \lambda(0, a_2, b_1) + (1 - \lambda)(0, a_1, b_2).$$

It is $\lambda \in [0, 1]$ since

$$\lambda < 0 \Rightarrow a = \lambda a_2 + (1 - \lambda)a_1 < a_1 + (1 - \lambda)a_1 = a_1 \qquad \lightning$$

and

$$\lambda > 1 \Rightarrow a = \lambda a_2 + (1 - \lambda)a_1 > a_2 + (1 - \lambda)a_2 = a_2. \qquad \lightning$$

Further, with Lemma 4.1.23 it follows

$$a_2 = b_1 + 1 \quad \text{and} \quad b_2 = a_1 + \delta.$$

Together with $a < b + 1$ and $b < a + \delta$ it follows $\lambda \in (0, 1)$. This is in contradiction to the result of Lemma 4.1.29. Thus, $a \geq b + 1$ and $b \geq a + \delta$ as needed.

- Case $x = 1$:

$$\begin{aligned} a &= (x + a - b) - 1 + b \leq 1 - 1 + b = b \\ b + 1 &= a + (x - a + b) \leq a + \delta \end{aligned}$$

Altogether, $(x, a, b) \in A$.

(ii) First, it is shown that every real convex combination of $(0, \alpha_1, \beta_1)$ and $(0, \alpha_2, \beta_2)$ is not an element of $A$:

With $\lambda \in (0, 1)$ it follows

$$\begin{aligned}[\lambda\alpha_1 + (1-\lambda)\alpha_2] &= \lambda(\beta_1 + 1) + (1-\lambda)(\beta_2 - \delta) \\ &= \lambda\beta_1 + (1-\lambda)\beta_2 - \delta + \lambda(\delta + 1) \\ &< \lambda\beta_1 + (1-\lambda)\beta_2 - \delta + \delta + 1 \\ &= [\lambda\beta_1 + (1-\lambda)\beta_2] + 1\end{aligned}$$

and

$$\begin{aligned}[\lambda\beta_1 + (1-\lambda)\beta_2] &= \lambda(\alpha_1 - 1) + (1-\lambda)(\alpha_2 + \delta) \\ &= \lambda\alpha_1 + (1-\lambda)\alpha_2 + \delta - \lambda(\delta + 1) \\ &< [\lambda\alpha_1 + (1-\lambda)\alpha_2] + \delta.\end{aligned}$$

Thus, $\lambda(0, \alpha_1, \beta_1) + (1-\lambda)(0, \alpha_2, \beta_2) \notin A$.

For both cases $\alpha_1 = \alpha_2$ and $\hat{r} < |\alpha_2 - \alpha_1|$ a real convex combination which lies in $\mathbb{Z}^2$ can be created:

- Case $\alpha_1 = \alpha_2$:
  It follows

  $$\beta_1 = \alpha_1 - 1 = \alpha_2 - 1 = \beta_2 - \delta - 1 \leq \beta_2 - 2.$$

  Then $(0, \alpha, \beta) := (0, \alpha_1, \beta_1 + 1) \in \mathbb{Z}^3$ is a real convex combination of $(0, \alpha_1, \beta_1)$ and $(0, \alpha_2, \beta_2)$.
- Case $\hat{r} < |\alpha_2 - \alpha_1|$:
  Lemma 4.1.29 provides a real convex combination $(0, \alpha, \beta) \in \mathbb{Z}^3$ of $(0, \alpha_1, \beta_1)$ and $(0, \alpha_2, \beta_2)$.

Therefore, there is no convex set $C \subseteq \mathbb{R}^3$ with $A = C \cap \mathbb{Z}^3$.

□

There is one last case left which is approached in the following theorem. If $F$ contains only points with either $s_k \geq s_{k'} + 1$ or only points with $s_{k'} \geq s_k + d_k$, then a representation by linear inequalities is possible.

**Theorem 4.1.31**
*Let $\delta \in \mathbb{N}$, $a_1$, $a_2$, $b_1$, $b_2 \in \mathbb{Z}$ with $a_1 \leq a_2$, $b_1 \leq b_2$ and $\mathbb{A} := [a_1, a_2] \cap \mathbb{Z}$, $\mathbb{B} := [b_1, b_2] \cap \mathbb{Z}$. Further, let $H := \max\{b_2 - a_1, a_2 - b_1\}$, $\mathbb{X} := \{0,1\} \times \mathbb{A} \times \mathbb{B}$ and*

$$\begin{aligned} A := \{(x,a,b) \in \mathbb{X} : (x = 0 \wedge (a \geq b+1 \vee b \geq a+\delta))\vee \\ (x = 1 \wedge (a \leq b \wedge b+1 \leq a+\delta))\}. \end{aligned}$$

*(i) If $A \cap \{(0,a,b) : a \geq b+1\} = \emptyset$, then*

$$\begin{aligned} A = \{(x,a,b) \in \mathbb{X} : (a+\delta-b-(H+\delta)x \leq 0)\wedge \\ (b+1-(a+\delta)+(H+\delta+1)(x-1) \leq 0)\}. \end{aligned}$$

*(ii) If $A \cap \{(0,a,b) : b \geq a+\delta\} = \emptyset$, then*

$$\begin{aligned} A = \{(x,a,b) \in \mathbb{X} : (b+1-a-(H+1)x \leq 0)\wedge \\ (a-b+H(x-1) \leq 0)\}. \end{aligned}$$

*Proof*
Since (ii) can be shown like (i), only the proof for (i) is demonstrated. With $A \cap \{(0,a,b) : a \geq b+1\} = \emptyset$ it follows $a \leq b$ for all $(x,a,b) \in A$ and $A$ can be rewritten as

$$A = \{(x,a,b) \in \mathbb{X} : (x = 1 \vee b \geq a+\delta) \wedge (x = 0 \vee b+1 \leq a+\delta)\}.$$

The claim follows with Theorem 4.1.8. $\square$

Theorems 4.1.28, 4.1.30 and 4.1.31 characterize in which cases new variables have to be introduced and in which cases Expression (4.1.12) can be modeled with linear inequalities. For the second case the according linear inequalities are already presented in Theorems 4.1.30 and 4.1.31.
For the cases where $s_k$, $s_{k'}$ and $p_{k,k'}$ are not sufficient to describe the feasible region two new sets of variables

$$ss_{k,k'} \in \{0,1\} \quad \text{and} \quad es_{k,k'} \in \{0,1\}$$

for $k, k' \in K$ are introduced. It is $ss_{k,k'} = 1$ if $j_k$ starts before $j_{k'}$ and (like before) $es_{k,k'} = 1$ if $j_k$ ends before $j_{k'}$ starts. Thus,

$$ss_{k,k'} = 1 \Leftrightarrow s_k \leq s_{k'} \quad \text{and} \quad es_{k,k'} = 1 \Leftrightarrow s_k + d_k \leq s_{k'}.$$

With Corollary 4.1.8 and the integrity of the variables this can be modeled as

$$\begin{aligned} s_{k'} - s_k - (H+1)ss_{k,k'} + 1 &\leq 0 \qquad (4.1.16)\\ s_k - s_{k'} + H(ss_{k,k'} - 1) &\leq 0 \\ s_{k'} - (s_k + d_k) - (H+1)es_{k,k'} + 1 &\leq 0 \\ (s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) &\leq 0, \qquad (4.1.17) \end{aligned}$$

where $H \in \mathbb{R}$ must be large enough.
The job $j_k$ is active at the starting time $s_{k'}$ if it starts before $s_{k'}$ but does not end before $s_{k'}$. Thus, $p_{k,k'}$ is connected to $ss_{k,k'}$ and $es_{k,k'}$ as follows:

$$p_{k,k'} = 1 \Leftrightarrow ss_{k,k'} = 1 \wedge es_{k,k'} = 0 \qquad (4.1.18)$$

To linearize this expression the following corollary is given.

**Corollary 4.1.32**
*Let* $\mathbb{X} := \{0,1\}^3$. *Then*

$$\begin{aligned} &\{(x_1, x_2, y) \in \mathbb{X} : (x_1 = 1 \wedge x_2 = 0) \Leftrightarrow y = 1\} \\ = &\{(x_1, x_2, y) \in \mathbb{X} : (2y - x_1 + x_2 \leq 1) \wedge (y - x_1 + x_2 \geq 0)\}. \end{aligned}$$

*Proof*
Since $[(x_1 = 1 \wedge x_2 = 0) \Leftrightarrow y = 1] \Leftrightarrow [(x_1 = 0 \vee x_2 = 1) \Leftrightarrow y = 0]$ holds, Lemmas 4.1.9 and 4.1.10 can be applied with $\tau_1 = \upsilon = 0$ and $\tau_2 = 1$. □

Using this result, Expression (4.1.18) can be formulated as:

$$2p_{k,k'} - ss_{k,k'} + es_{k,k'} \leq 1 \qquad (4.1.19)$$

$$p_{k,k'} - ss_{k,k'} + es_{k,k'} \geq 0 \qquad (4.1.20)$$

Similar to the last section, Equations (4.1.19) and (4.1.20) can be tightened in such a way that all $p$ variables become redundant. To see this the following results are given in a short informal notation. The meaning of the terms and variables should be clear from their use above.

**Lemma 4.1.33**
$es_{k,k'} - ss_{k,k'} \leq 0$.

*Proof*
Suppose $1 = es_{k,k'} > ss_{k,k'} = 0$. Then $s_k + d_k \leq s_{k'}$ and $s_{k'} + 1 \leq s_k$ by Constraints (4.1.17) and (4.1.16) and therefore $s_k + d_k < s_k$. This is in contradiction to the definition. Thus, $es_{k,k'} \leq ss_{k,k'}$ holds. $\square$

**Lemma 4.1.34**
*Equations* (4.1.19) *and* (4.1.20) *can be substituted by* $p_{k,k'} - ss_{k,k'} + es_{k,k'} = 0$.

*Proof*
By adding the inequality in Lemma 4.1.33 to Equation (4.1.19)

$$2p_{k,k'} - 2ss_{k,k'} + 2es_{k,k'} \leq 1$$

follows. Since all variables are integer, this resolves to

$$p_{k,k'} - ss_{k,k'} + es_{k,k'} \leq 0. \tag{4.1.21}$$

In combination with Equation (4.1.20) the claim is proven. $\square$

Therefore, the $p$, $ss$ and $es$ variables are in direct dependency and thus the $p$ variables in Equation (4.1.11) can be substituted as follows.

**Corollary 4.1.35**
*Equation* (4.1.11) *is equivalent to*

$$\sum_{k \in K} ss_{k,k'} - es_{k,k'} \leq z$$

*for all* $k' \in K$.

*Proof*
The equation in Lemma 4.1.34 is equivalent to $p_{k,k'} = ss_{k,k'} - es_{k,k'}$. Substitution of $p_{k,k'}$ in Equation (4.1.11) proofs the claim. $\square$

Therefore, all $p$ variables can be omitted and only the constraints

$$\begin{aligned}
s_{k'} - s_k - (H+1)ss_{k,k'} + 1 &\leq 0 \\
s_k - s_{k'} + H(ss_{k,k'} - 1) &\leq 0 \\
s_{k'} - (s_k + d_k) - (H+1)es_{k,k'} + 1 &\leq 0 \\
(s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) &\leq 0
\end{aligned}$$

for all $k, k' \in K$ and

$$\sum_{k \in K} ss_{k,k'} - es_{k,k'} \leq z$$

for all $k' \in K$ describe the feasible region.

This leads to the following definition, where with the same arguments as before the integrality conditions for $s$ and $z$ are omitted.

**Definition 4.1.36**
*Let $\mathcal{I} := (K, M, T, \hat{t}, r, c, f)$ be an instance of MJAP, where $s_k^{\min}$ and $s_k^{\max}$ are the bounds for the starting time and $d_k$ the duration of movable job $k \in K$. Define $H := \max_{k,k' \in K}\{(s_k^{\max} + d_k) - s_{k'}^{\min}\}$. The problem to find a set of corresponding jobs, where the size of the largest overlapping set is minimal, is called MinOSlin$_{\mathcal{I}}$ and is defined as the following ILP:*

$$\begin{aligned}
&\min\ z \\
&\quad s.t. \\
s_{k'} - s_k - (H+1)ss_{k,k'} + 1 &\leq 0 && \forall k, k' \in K \\
s_k - s_{k'} + H(ss_{k,k'} - 1) &\leq 0 && \forall k, k' \in K \\
s_{k'} - (s_k + d_k) - (H+1)es_{k,k'} + 1 &\leq 0 && \forall k, k' \in K \\
(s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) &\leq 0 && \forall k, k' \in K \\
\sum_{k \in K} ss_{k,k'} - es_{k,k'} &\leq z && \forall k' \in K
\end{aligned}$$

$$\begin{aligned}
s_k &\in [s_k^{\min}, s_k^{\max}] && \forall k \in K \\
z &\in \mathbb{R} \\
ss_{k,k'} &\in \{0,1\} && \forall k, k' \in K \\
es_{k,k'} &\in \{0,1\} && \forall k, k' \in K
\end{aligned}$$

In the implementation of MinOSlin$_{\mathcal{I}}$ in Section 4.3 the variables $ss$ and $es$ are only created if they are not predetermined. If for example $s_k^{\max} > s_{k'}^{\min}$, then $ss_{k,k'}$ must be 0 and is not created. Further, $ss$ and $es$ are not created if a formulation with $p$ is possible. In this case the appropriate inequalities from Theorems 4.1.30 and 4.1.31 are used.

## 4.2 Compact formulation

In the last section a heuristic to solve MJAP was presented. In this section a heuristic for JAP is developed to tackle instances which are too large to be solved with JAPlin$_{\mathcal{I}}$. The idea is to substitute all machines of one type by one single object and in this way to create a more compact formulation. Thus, jobs get assigned to machine types instead of machines. Motivated by Theorem 4.1.22 the number of machines of one type can be determined by calculating the largest overlapping set of the assigned jobs. To not exceed the number of available machines the size of the largest overlapping set is bounded by $|\hat{t}^{-1}(t)|$ for each machine type $t$. The same approach is also applied to derive another heuristic for MJAP in Section 4.2.1. The compact formulation for JAP is given in the following definition.

**Definition 4.2.1** (Compact Convex Job Assignment Problem)
*Let $J$ be a finite set of jobs, where $d_j$ is the duration of job $j \in J$. Further, let $M$ and $T$ be finite sets, $\hat{t} : M \to T$, $r \in \{0,1\}^{J \times T}$, $c_0, c_1, c_2 \in \mathbb{R}_0^+$ with $c_1 \le c_2$ and $f \in \mathbb{N}_0$.*
*The problem to find an assignment $a : J \to T$ with $r_{j,a(j)} = 1$ for all $j \in J$ and $\rho_{a^{-1}(t)} \le |\hat{t}^{-1}(t)|$ for all $t \in T$, where*

$$c_0 \sum_{t \in T} \rho_{a^{-1}(t)} + \sum_{t \in T} \max \left\{ c_1 \sum_{j \in a^{-1}(t)} d_j, c_2 \sum_{j \in a^{-1}(t)} d_j - \rho_{a^{-1}(t)} f (c_2 - c_1) \right\}$$

*is minimal, is called* Compact Convex Job Assignment Problem *(CJAP).*

In this notation $\rho_{a^{-1}(t)}$ represents the number of needed machines of type $t$ since this is the size of the largest overlapping set of the jobs assigned to $t$. The number of available machines of type $t$ must not be exceeded. Thus $\rho_{a^{-1}(t)} \le |\hat{t}^{-1}(t)|$. Further, the production limit $f$ has to be multiplied by $\rho_{a^{-1}(t)}$ since it is provided for each machine. At this point it becomes apparent that CJAP is just a heuristic to solve JAP. If the assigned jobs to a machine type $t$ can be evenly distributed among the $\rho_{a^{-1}(t)}$ machines, the solution value of CJAP should be very close to the corresponding solution value of JAP. However, if these jobs cannot be evenly distributed, e.g. if the durations of the jobs have a high variance and the durations of some jobs are large compared to $f$, great differences can arise. This is demonstrated in the following example:

Let $j_1 = (0, 1)$ and $j_2 = (0, 9)$ be two jobs starting at time 0 with the durations 1 and 9. There are two machines $m_1$ and $m_2$ both of type $t_1$. For simplicity reasons the restriction matrix $r$ is 1 in each component and $c_0 = 0$. Further, let $f = 5$ with $c_1 \leq c_2$. For CJAP the only solution is to assign $j_1$ and $j_2$ to $t_1$. Since $j_1$ and $j_2$ overlap, it is $\rho_{a^{-1}(t_1)} = 2$. This results in a solution value of

$$\max\{c_1 \cdot (1 + 9), c_2 \cdot (1 + 9) - 2 \cdot 5 \cdot (c_2 - c_1)\} = 10c_1.$$

For JAP the only solution is to assign $j_1$ to $m_1$ and $j_2$ to $m_2$ (or vice versa since $m_1$ and $m_2$ are exchangeable). This results in a solution value of

$$\begin{aligned}
&\max\{c_1 \cdot 1, c_2 \cdot 1 - 5 \cdot (c_2 - c_1)\} + \max\{c_1 \cdot 9, c_2 \cdot 9 - 5 \cdot (c_2 - c_1)\} \\
&= c_1 + 4c_2 + 5c_1 \\
&= 6c_1 + 4c_2 \\
&\geq 10c_1.
\end{aligned}$$

Thus, depending on the difference between $c_1$ and $c_2$ the solution values for JAP and CJAP can vary strongly. How much the heuristic solution can differ from the optimal solution, depending on the input parameters, is a field of future research.

Furthermore, another term in Definition 4.2.1 is conspicuous. To get the total fixed costs for each machine type the number of needed machines is multiplied by $c_0$, which gives $c_0 \sum_{t \in T} \rho_{a^{-1}(t)}$. At the first glance it seems to be easier to simply calculate $c_0 \rho_J$. However, in general and also in the case of an optimal assignment $\sum_{t \in T} \rho_{a^{-1}(t)} = \rho_J$ may not hold:

Consider the following example with the four jobs $j_1 = (0, 4)$, $j_2 = (0, 4)$, $j_3 = (0, 2)$ and $j_4 = (2, 2)$. Thus, three jobs start at time 0, two of which have the duration 4 and one has the duration 2, and one job with duration 2 starts at time 2. Further, there are 3 machines of type $t_1$ and 3 machines of type $t_2$, where only $t_1$ is allowed to process $j_1$ and only $t_2$ is allowed to process $j_2$. Because of the symmetry $j_1$ and $j_2$, as well as $j_3$ and $j_4$ are interchangeable. This leads to only two feasible assignments:

$$\begin{array}{c} \{j_1, j_3\} \mapsto t_1 \\ \{j_2, j_4\} \mapsto t_2 \end{array} \quad \text{and} \quad \begin{array}{c} \{j_1, j_3, j_4\} \mapsto t_1 \\ \{j_2\} \mapsto t_2 \end{array}$$

Further, let $f = 3$ and $c_0 \ll c_1 \ll c_2$. Thus, $c_1$ is much larger than $c_0$ and $c_2$ is much larger than $c_1$. Since

$$\rho_{\{j_1,j_3\}} = \rho_{\{j_1,j_3,j_4\}} = \rho_{\{j_2,j_4\}} = \rho_{\{j_2,j_3,j_4\}} = 2,$$

it is optimal to assign $j_1$ and $j_3$ to $t_1$, and $j_2$ and $j_4$ to $t_2$ because the total job duration on both machine types equals the production limit $(4 + 2 = 2 \cdot 3 = \rho_{a^{-1}(t_1)} f)$ and consequently the variable costs are $(2 + 4)c_1$. If $j_1$, $j_3$ and $j_4$ would be assigned to machine type $t_1$, then $2 + 2 + 4 > 2 \cdot 3 = \rho_{a^{-1}(t_1)} f$ and the variable costs would be $2c_2 + (2 + 4)c_1$. Since $c_1 \ll c_2$, these solutions are not preferred.
Therefore, in the optimal solution $j_1$ and $j_3$ are assigned to $t_1$, and $j_2$ and $j_4$ are assigned to $t_2$. However, it is $\rho_J = 3$ and $\rho_{a^{-1}(t_1)} + \rho_{a^{-1}(t_2)} = 2 + 2 = 4$. Thus, even for an optimal assignment $\sum_{t \in T} \rho_{a^{-1}(t)} \neq \rho_J$ may not hold and $c_0 \sum_{t \in T} \rho_{a^{-1}(t)}$ has to be calculated.

As an insight into the complexity of CJAP the following theorem is given.

**Theorem 4.2.2**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves CJAP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|J|,|M|,|T|,f)}\right)$ for some polynomial* poly : $\mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 1}$. *This is true even if $|M| = 2$, $|T| = 2$, $r_{j,t} = 1$ for all $j \in J$ and $t \in T$, $c_0 = 0$ and $c_1 = 1$.*

*Proof*
The proof is analogous to the proof of Theorem 4.1.3 except that the assignment function $a$ assigns jobs to machine types and not to machines. $\square$

The ILP to solve an instance $\mathcal{I}$ of CJAP is very similar to JAPlin$_{\mathcal{I}}$. The variables $x_{j,m}$ are substituted by

$$x_{j,t} \in \{0,1\}$$

for $j \in J$ and $t \in T$, where $x_{j,t} = 1$ encodes „job $j$ is processed by a machine of type $t$“. As before each job must be assigned to exactly one machine type. This is ensured by

$$\sum_{t \in T} x_{j,t} = 1$$

for all $j \in J$. A new set of integer variables

$$z_t \in \mathbb{N}_0$$

for $t \in T$ is introduced. These variables count the number of needed machines of type $t \in T$ (which is $\rho_{a^{-1}(t)}$) and therefore are bounded by $|\hat{t}^{-1}(t)|$. With Lemma 4.0.5 Equation (4.1.1) can be replaced by

$$\sum_{j \in O} x_{j,t} \leq z_t$$

for all $t \in T$ and all overlapping sets $O \in O_J$. The target function becomes

$$c_0 \sum_{t \in T} z_t + \sum_{t \in T} \max \left\{ c_1 \sum_{j \in J} d_j x_{j,t}, c_2 \sum_{j \in J} d_j x_{j,t} - z_t f(c_2 - c_1) \right\},$$

which can be linearized like before. A set of new variables

$$y_t \in \mathbb{R}$$

is introduced for $t \in T$. These form an upper bound for the maximum expression in the second summand. This is ensured by the constraints

$$c_1 \sum_{j \in J} d_j x_{j,t} \leq y_t \quad \text{and} \quad c_2 \sum_{j \in J} d_j x_{j,t} - z_t f(c_2 - c_1) \leq y_t$$

for all $t \in T$.

Thus, the cost function becomes

$$c_0 \sum_{t \in T} z_t + \sum_{t \in T} y_t.$$

The complete ILP is presented in the following definition.

**Definition 4.2.3**
*Let $\mathcal{I} := (J, M, T, \hat{t}, r, c, f)$ be an instance of CJAP, where $d_j$ is the duration of job $j \in J$. The problem CJAPlin$_\mathcal{I}$ is defined as the following ILP:*

$$\min \; c_0 \sum_{t \in T} z_t + \sum_{t \in T} y_t$$

$$\text{s.t.}$$

$$\begin{aligned}
\sum_{t \in T} x_{j,t} &= 1 && \forall j \in J \\
x_{j,t} &\leq r_{j,t} && \forall j \in J, t \in T \\
c_1 \sum_{j \in J} d_j x_{j,t} &\leq y_t && \forall t \in T \\
c_2 \sum_{j \in J} d_j x_{j,t} - z_t f (c_2 - c_1) &\leq y_t && \forall t \in T \\
\sum_{j \in O} x_{j,t} &\leq z_t && \forall t \in T, O \in O_J \\
z_t &\leq |\hat{t}^{-1}(t)| && \forall t \in T
\end{aligned}$$

$$\begin{aligned}
x_{j,t} &\in \{0,1\} && \forall j \in J, t \in T \\
y_t &\in \mathbb{R} && \forall t \in T \\
z_t &\in \mathbb{N}_0 && \forall t \in T
\end{aligned}$$

### 4.2.1 Movable jobs

As for JAP a compact formulation can be introduced for the time window version MJAP. The mathematical description is presented in the following definition and is a composition of MJAP and CJAP.

**Definition 4.2.4** (Compact Convex Movable Job Assignment Problem)
*Let $K$ be a finite set of movable jobs, where $d_k$ is the duration of movable job $k \in K$. Further, let $M$ and $T$ be finite sets, $\hat{t} : M \to T$, $r \in \{0,1\}^{K \times T}$, $c_0, c_1, c_2 \in \mathbb{R}_0^+$ with $c_1 \leq c_2$ and $f \in \mathbb{N}_0$.*

*The problem to find a feasible set of corresponding jobs $J = \{j_k\}_{k \in K}$ for $K$ and an assignment $a : J \to T$ with $r_{k,a(j_k)} = 1$ for all $k \in K$ and $\rho_{a^{-1}(t)} \leq |\hat{t}^{-1}(t)|$ for all $t \in T$, where*

$$c_0 \sum_{t \in T} \rho_{a^{-1}(t)} + \sum_{t \in T} \max \left\{ c_1 \sum_{j \in a^{-1}(t)} d_j, c_2 \sum_{j \in a^{-1}(t)} d_j - \rho_{a^{-1}(t)} f(c_2 - c_1) \right\}$$

*is minimal, is called the* Compact Convex Movable Job Assignment Problem *(CMJAP).*

CMJAP does not admit a polynomial-time approximation algorithm with exponential approximation ratio. This is stated in the following theorem.

**Theorem 4.2.5**
*Unless $\mathbb{P} = \mathbb{NP}$, there is no polynomial-time approximation algorithm that solves CMJAP with approximation ratio in $\mathcal{O}\left(2^{\text{poly}(|K|,|M|,|T|,f)}\right)$ for some polynomial* $\text{poly} : \mathbb{R}^3_{\geq 0} \to \mathbb{R}_{\geq 1}$. *This is true even if $|M| = 2$, $|T| = 2$, $r_{k,t} = 1$ for all $k \in K$ and $t \in T$, $c_0 = 0$ and $c_1 = 1$.*

*Proof*
The proof is analogous to the proof of Theorem 4.1.6 in combination with Theorem 4.2.2 except that the function $a$ assigns jobs to machine types and not to machines. □

An ILP for CMJAP is created as a mixture of MJAPlin$_\mathcal{I}$ and CJAPlin$_\mathcal{I}$. The same $x$ and $z$ variables as in CJAPlin$_\mathcal{I}$ are used except that $x$ is not indexed by jobs anymore (index $j$) but by movable jobs (index $k$). Furthermore, like for MJAPlin$_\mathcal{I}$, the variables $ss$ and $es$ encode if a job starts or ends before another job starts. In Section 4.1.2 the constraints $\sum_{k \in K} ss_{k,k'} - es_{k,k'} \leq z$ are used to determine the size of the overlapping set at time $s_{k'}$. Since for CMJAP only those jobs assigned to a machine type contribute to an overlapping set, these constraints change to

$$\sum_{k \in K} x_{k,t}(ss_{k,k'} - es_{k,k'}) \leq z_t \tag{4.2.1}$$

for all $t \in T$.

Obviously, this is no linear expression. Thus, two new sets of binary variables $xss$ and $xes$ with

$$xss_{k,k',t} = x_{k,t}ss_{k,k'} \in \{0,1\} \tag{4.2.2}$$

and

$$xes_{k,k',t} = x_{k,t}es_{k,k'} \in \{0,1\} \tag{4.2.3}$$

for all $k, k' \in K$ and $t \in T$ are introduced. The corresponding logical expressions are

$$xss_{k,k',t} = 1 \Leftrightarrow x_{k,t} = 1 \wedge ss_{k,k'} = 1 \quad \text{and} \quad xes_{k,k',t} = 1 \Leftrightarrow x_{k,t} = 1 \wedge es_{k,k'} = 1.$$

Equations (4.2.2) and (4.2.3) are still nonlinear but due to the integrity of the variables a linear representation exists. This representation is given in the following corollary.

**Corollary 4.2.6**
*Let* $\mathbb{X} := \{0,1\}^3$. *Then*

$$\begin{aligned} & \{(x_1, x_2, y) \in \mathbb{X} : (x_1 = 1 \wedge x_2 = 1) \Leftrightarrow y = 1\} \\ = \ & \{(x_1, x_2, y) \in \mathbb{X} : (2y - x_1 - x_2 \le 0) \wedge (x_1 + x_2 - y \le 1)\}. \end{aligned}$$

*Proof*
Since $[(x_1 = 1 \wedge x_2 = 1) \Leftrightarrow y = 1] \Leftrightarrow [(x_1 = 0 \vee x_2 = 0) \Leftrightarrow y = 0]$ holds, Lemmas 4.1.9 and 4.1.10 can be applied with $\tau_1 = \tau_2 = \upsilon = 0$. □

Therefore, Equations (4.2.2) and (4.2.3) resolve to

$$\begin{aligned} 2xss_{k,k',t} - x_{k,t} - ss_{k,k'} &\le 0 \\ x_{k,t} + ss_{k,k'} - xss_{k,k',t} &\le 1 \\ 2xes_{k,k',t} - x_{k,t} - es_{k,k'} &\le 0 \\ x_{k,t} + es_{t,u} - xes_{k,k',t} &\le 1 \end{aligned}$$

for all $k, k' \in K$ and $t \in T$.

Equation (4.2.1) can be rewritten as

$$\sum_{k \in K} xss_{k,k',t} - xes_{k,k',t} \leq z_t$$

for all $k' \in K$ and $t \in T$. This leads to the following ILP for CMJAP.

**Definition 4.2.7**
*Let $\mathcal{I} := (K, M, T, \hat{t}, r, c, f)$ be an instance of CMJAP, where $s_k^{\min}$ and $s_k^{\max}$ are the bounds for the starting time and $d_k$ the duration of movable job $k \in K$. Define $H := \max_{k,k' \in K}\{(s_k^{\max} + d_k) - s_{k'}^{\min}\}$. The problem CMJAPlin$_\mathcal{I}$ is defined as the following ILP:*

$$\begin{aligned}
\min\ c_0 \sum_{t \in T} z_t + \sum_{t \in T} y_t & \\
s.t. & \\
\sum_{t \in T} x_{k,t} = 1 & \qquad \forall k \in K \\
x_{j,t} \leq r_{j,t} & \qquad \forall j \in J, t \in T \\
c_1 \sum_{k \in K} d_k x_{k,t} \leq y_t & \qquad \forall t \in T \\
c_2 \sum_{k \in K} d_k x_{k,t} - z_t f(c_2 - c_1) \leq y_t & \qquad \forall t \in T \\
s_{k'} - s_k - (H+1) ss_{k,k'} + 1 \leq 0 & \qquad \forall k, k' \in K \\
s_k - s_{k'} + H(ss_{k,k'} - 1) \leq 0 & \qquad \forall k, k' \in K \\
s_{k'} - (s_k + d_k) - (H+1) es_{k,k'} + 1 \leq 0 & \qquad \forall k, k' \in K \\
(s_k + d_k) - s_{k'} + H(es_{k,k'} - 1) \leq 0 & \qquad \forall k, k' \in K \\
2xss_{k,k',t} - x_{k,t} - ss_{k,k'} \leq 0 & \qquad \forall k, k' \in K, t \in T \\
x_{k,t} + ss_{k,k'} - xss_{k,k',t} \leq 1 & \qquad \forall k, k' \in K, t \in T \\
2xes_{k,k',t} - x_{k,t} - es_{k,k'} \leq 0 & \qquad \forall k, k' \in K, t \in T \\
x_{k,t} + es_{t,u} - xes_{k,k',t} \leq 1 & \qquad \forall k, k' \in K, t \in T \\
\sum_{k \in K} xss_{k,k',t} - xes_{k,k',t} \leq z_t & \qquad \forall k' \in K, t \in T \\
z_t \leq |\hat{t}^{-1}(t)| & \qquad \forall t \in T
\end{aligned}$$

$$\begin{aligned}
x_{k,t} &\in \{0,1\} && \forall k \in K, t \in T\\
y_t &\in \mathbb{R} && \forall t \in T\\
s_k &\in [s_k^{\min}, s_k^{\max}] && \forall k \in K\\
z_t &\in \mathbb{N}_0 && \forall t \in T\\
ss_{k,k'} &\in \{0,1\} && \forall k,k' \in K\\
es_{k,k'} &\in \{0,1\} && \forall k,k' \in K\\
xss_{k,k',t} &\in \{0,1\} && \forall k,k' \in K, t \in T\\
xes_{k,k',t} &\in \{0,1\} && \forall k,k' \in K, t \in T
\end{aligned}$$

### 4.2.2 Assignment of jobs

Until now only the assignment of jobs to machine types was covered. The assignment of jobs to single machines is left. To approach this problem a small ILP, which has to be solved for every machine type, is designed in the following. The goal is to distribute the production times (i.e. durations) evenly between the machines of one type. In this way, the production limit $f$ is exhausted first and hopefully only $c_1$ has to be paid per time unit.
The program for the CJAP case is introduced in the following. Let $t$ be some machine type. The variable $z_t$, as part of a solution of CJAPlin$_{\mathcal{I}}$, denotes the number of machines which have to be provided of this type. In the following definition these machines are denoted by $M_t = \{1, \ldots, z_t\}$. Further, $J_t$ holds all jobs that have to be processed by these machines. The binary variables $u$ encode the assignment of jobs to machines ($u$ is chosen to not get confused with the $x$ variables of the solution of CJAPlin$_{\mathcal{I}}$). Thus,

$$\sum_{j \in J_t} d_j u_{j,m}$$

is the total processing duration of machine $m$. To create an evenly distributed assignment the maximum of all total processing times is minimized. Thus, a variable $v$ is introduced, which denotes the maximal processing time of all machines. The constraints

$$\sum_{j \in J_t} d_j u_{j,m} \leq v$$

ensure that $v$ is an upper bound for this maximum. Therefore, if $v$ is minimized, the maximal processing time is minimized as needed. Additionally, the jobs on one machine are not allowed to overlap. This is modeled by

$$\sum_{j\in O} u_{j,m} \leq 1$$

for all $O \in O_{J_t}$. The complete ILP is presented in the following definition.

**Definition 4.2.8**
*Let $\mathcal{I} := (J, M, T, \hat{t}, r, c, f)$ be an instance of CJAP. Further, let $x$ and $z$ be the $x$ and $z$ part of a solution vector of CJAPlin$_\mathcal{I}$ and $t \in T$. Define $M_t := \{1, \dots, z_t\}$ and $J_t := \{j \in J : x_{j,t} = 1\}$. The ILP to solve the machine assignment problem for CJAP is called CJAP$_{x,z,t}$lin$_\mathcal{I}$ and defined as follows:*

$$\begin{aligned}
&\min\ v \\
&\quad s.t. \\
&\sum_{m\in M_t} u_{j,m} = 1 && \forall j \in J_t \\
&\sum_{j\in O} u_{j,m} \leq 1 && \forall m \in M_t, O \in O_{J_t} \\
&\sum_{j\in J_t} d_j u_{j,m} \leq v && \forall m \in M_t \\
&u_{j,m} \in \{0,1\} && \forall j \in J_t, m \in M_t \\
&v \in \mathbb{R}
\end{aligned}$$

To find a solution for JAP first CJAPlin$_\mathcal{I}$ is solved. This produces the solution vectors $x$ and $z$. Afterwards, for each machine type $t$ the machine assignment problem CJAP$_{x,z,t}$lin$_\mathcal{I}$ is solved. This creates a valid job to machine assignment for JAP, which can be evaluated with respect to the target function of JAP.

Also for CMJAP an assignment problem which assigns jobs to machines has to be solved for each machine type. This is done analogously as for CJAP with one little distinction. The starting times of the movable jobs are determined by CMJAPlin$_\mathcal{I}$ in the vector $s$. Thus, the set of jobs for machine type $t$ depends also on $s$. It is denoted as $J_{s,t} := \{(s_k, d_k) : x_{k,t} = 1, k \in K\}$. As before, this leads to the following definition.

**Definition 4.2.9**
*Let $\mathcal{I} := (K, M, T, \hat{t}, r, c, f)$ be an instance of CMJAP, where $d_k$ is the duration of movable job $k \in K$. Further, let $x$, $s$ and $z$ be the $x$, $s$ and $z$ part of a solution vector of CMJAPlin$_\mathcal{I}$ and $t \in T$. Define $M_t := \{1, \dots, z_t\}$ and $J_{s,t} := \{(s_k, d_k) : x_{k,t} = 1, k \in K\}$. The ILP to solve the machine assignment problem for CMJAP is called CMJAP$_{x,s,z,t}$lin$_\mathcal{I}$ and defined as follows:*

$$
\begin{aligned}
&\min\ v \\
&\quad s.t. \\
&\sum_{m \in M_t} u_{j,m} = 1 && \forall j \in J_{s,t} \\
&\sum_{j \in O} u_{j,m} \leq 1 && \forall m \in M_t, O \in O_{J_{s,t}} \\
&\sum_{j \in J_{s,t}} d_j u_{j,m} \leq v && \forall m \in M_t \\
&u_{j,m} \in \{0,1\} && \forall j \in J_{s,t}, m \in M_t \\
&v \in \mathbb{R}
\end{aligned}
$$

To find a solution for MJAP first CMJAPlin$_\mathcal{I}$ is solved. This produces the solution vectors $x$, $s$ and $z$. The sets of jobs $J_{s,t}$ are created for each machine type $t$ (note that the union $\bigcup_{t \in T} J_{s,t}$ forms a valid set of jobs for MJAP). After that, for each machine type the machine assignment problem CMJAP$_{x,s,z,t}$lin$_\mathcal{I}$ is solved. This creates a valid job to machine assignment for MJAP, which can be evaluated with respect to the target function of MJAP.

Altogether, besides solving an instance $\mathcal{I}$ of MJAP optimally with MJAPlin$_\mathcal{I}$, there are three ways to get a heuristic solution:

- solve CMJAPlin$_\mathcal{I}$ $\rightarrow$ solve CMJAP$_{x,s,z,t}$lin$_\mathcal{I}$
- solve MinOSlin$_\mathcal{I}$ $\rightarrow$ solve JAPlin$_\mathcal{I}$
- solve MinOSlin$_\mathcal{I}$ $\rightarrow$ solve CJAPlin$_\mathcal{I}$ $\rightarrow$ solve CJAP$_{x,z,t}$lin$_\mathcal{I}$

All these solving methods are tested in the next section.

## 4.3 Numerical results

To test the algorithms developed in the last sections instances of MJAP with three different values for $|K|$ and $|T|$ are generated. Additionally, three different so-called „diversity factors“ are tested, which is explained later. As test instances for JAP the generated instances of MJAP are used, except that the movable jobs are substituted by jobs with fixed starting time $s^{\min}$ and duration $d$.

Since JAP and MJAP get harder to solve if jobs are overlapping more often, instances with smaller and larger overlapping sets are created. For that, a diversity factor is introduced. This factor scales the (integer) interval in which the starting times are randomly chosen. This means, if the diversity factor is large, also the interval is large. Therefore, jobs do not tend to overlap. If the factor and hence the interval are small, the starting times of the jobs are close together and the jobs tend to overlap more often.
The maximal duration of the movable jobs in all instances is set to 10. Thus, for all $k \in K$ the duration $d_k \in \{1, 2, \ldots, 10\}$ is randomly chosen. The previously mentioned interval for the starting times is defined as

$$\{0, 1, \ldots, \lceil \text{diversity} \cdot (|K| - 1) \cdot 10 \rceil\}.$$

In this way for a diversity of 1.0 a set of $|K|$ movable jobs could be arranged one after another such that no two jobs overlap. For a smaller diversity factor the interval gets smaller and at some point jobs have to overlap. Nevertheless, since all values are chosen randomly, this also happens already for a diversity factor of 1.0. Still, if the diversity gets smaller the sizes of the overlapping sets increase.
For each $k \in K$ the lower bound for the starting time.

$$s^{\min} \in \{0, 1, \ldots, \lceil \text{diversity} \cdot (|K| - 1) \cdot 10 \rceil\}$$

and the upper bound

$$s^{\max} \in \left\{s^{\min}, s^{\min} + 1, \ldots, s^{\min} + 10\right\}$$

are chosen randomly. This means, each movable job may at most be moved by 10 time units. For JAP the set of jobs is defined as $J := \{(s_k^{\min}, d_k)\}_{k \in K}$.

The number of machines for each machine type is chosen as

$$\left\lceil \frac{\rho_J}{|T|-1} \right\rceil .$$

In this way enough machines should be provided to process all jobs since

$$|M| = \left\lceil \frac{\rho_J}{|T|-1} \right\rceil \cdot |T| > \rho_J .$$

The machine type function $\hat{t}$ is chosen canonical such that the number of machines is the same for each machine type. The restriction matrix $r$ is chosen randomly such that about 90% of the entries equal 1.
The production limit is defined as

$$f := \left\lceil \frac{\sum_{k\in K} d_k}{\rho_J} \right\rceil .$$

Like this, if all jobs can be distributed evenly, the production times of the machines should be around $f$. However, since $r$ restricts some job machine type combinations, some machines should exceed the production limit and some should require less time. This is wanted because otherwise the production limit would not have any impact on the solution. Finally, the cost vector is chosen as $(c_0, c_1, c_2) := (3f, 1, 2)$.
The ILP solver (Xpress) often detects infeasible instances very fast. If a generated instance is infeasible (for example, if $r$ is chosen such that no machine type is allowed for a job), it is generated randomly again.
It has to be mentioned that in the ILP implementations of JAPlin$_\mathcal{I}$, CJAPlin$_\mathcal{I}$ and CJAP$_{x,z,t}$lin$_\mathcal{I}$ not all constraints for $O \in O_J$ are created. For two overlapping sets $O, O' \in O_J$ with $O \subseteq O'$ it is

$$\sum_{j\in O} x_{j,m} \leq \sum_{j \in O'} x_{j,m} \leq 1.$$

Therefore, the constraint for $O$ is already induced by the constraint for $O'$. Therefore, the constraint for an overlapping set $O$ is only created if no other overlapping set $O'$ with $O \subseteq O'$ exists. Further, as already mentioned, in the implementations of MJAPlin$_\mathcal{I}$, MinOSlin$_\mathcal{I}$ and CMJAPlin$_\mathcal{I}$ the $ss$ and $es$ variables and the according constraints are only created if they are not predetermined.

To solve JAP two methods were developed. The first is to solve JAPlin$_{\mathcal{I}}$, which produces the optimal solution. The second is to solve CJAPlin$_{\mathcal{I}}$ and then CJAP$_{x,z,t}$lin$_{\mathcal{I}}$ for each machine type. This produces a heuristic solution. The results for both methods are noted in Table 4.1 (columns JAP and CJAP). Not only the runtime of the algorithms but also the solution values are given. Further, the results of a randomized algorithm are stated in the last column. The randomized algorithm tries random assignments of jobs to machines and returnes the best assignment. The algorithm is run for 100 times. To make the results comparable the runtime for each run is bounded by the time JAPlin$_{\mathcal{I}}$ needed to find the optimal solution (column JAP $\rightarrow$ time). Besides the ratio how often a feasible solution was found in the 100 runs, the average target value of these solutions is given. For the cases in which no solution was found at all no target value exists. This is denoted by „-“.
The most eye-catching detail in Table 4.1 are the target values of in the columns JAP and CJAP. Without exception the heuristic found not only the optimal solution but this especially in a fraction of the time that was needed to solve JAPlin$_{\mathcal{I}}$. This is rooted in the fact that the durations of the jobs just range from 1 to 10, which is small compared to $f$. Thus, the jobs can be evenly distributed among the machines and the bad case like the example after Definition 4.2.1 does not occur.
All instances can be solved by the heuristic in less than 0.3 seconds. However, the runtime for JAPlin$_{\mathcal{I}}$ and the results of the randomized algorithm strongly depend on $|J|$, $|T|$ and the diversity.
For $|J| = 50$ all instances can be solved optimally in about 1 second or less. The randomized algorithm nearly always finds a solution except for $|T| = 5$ and a diversity of 1.0 or 0.6. In these two cases a solution is only found in 95% or 14% of the 100 runs.
For $|J| = 100$ the runtime of JAPlin$_{\mathcal{I}}$ ranges up to about 15 seconds for $|T| = 8$ and a diversity of 0.6. Already for this number of jobs it gets hard for the randomized algorithm to find feasible solutions. For a diversity factor of 0.6 never a solution is found. For $|T| = 3$ and a diversity of 1.0 in 60% of the runs a solution is found, for a diversity of 0.8 just in 20%.
For $|J| = 200$ the randomized algorithm never finds a feasible solution and the runtime of JAPlin$_{\mathcal{I}}$ goes up to about 57 seconds for a diversity of 0.6 and 8 machine types. It is apparent that the runtime of JAPlin$_{\mathcal{I}}$ gets larger if the

| \|J\| | \|T\| | div. | JAP | | CJAP | | JAP rand. | |
|---|---|---|---|---|---|---|---|---|
| | | | time | target | time | target | ratio | target |
| 50 | 3 | 1.0 | 0.434 | 852 | 0.090 | 852 | 100% | 996.8 |
| | | 0.8 | 0.706 | 838 | 0.104 | 838 | 100% | 978.0 |
| | | 0.6 | 1.064 | 716 | 0.084 | 716 | 100% | 967.6 |
| | 5 | 1.0 | 0.312 | 896 | 0.091 | 896 | 97% | 1098.3 |
| | | 0.8 | 0.244 | 854 | 0.087 | 854 | 100% | 1047.3 |
| | | 0.6 | 0.458 | 826 | 0.106 | 826 | 14% | 896.0 |
| | 8 | 1.0 | 0.970 | 846 | 0.103 | 846 | 100% | 1292.1 |
| | | 0.8 | 0.568 | 836 | 0.112 | 836 | 100% | 1283.3 |
| | | 0.6 | 0.989 | 820 | 0.104 | 820 | 100% | 1353.8 |
| 100 | 3 | 1.0 | 2.355 | 1744 | 0.133 | 1744 | 60% | 2040.3 |
| | | 0.8 | 2.035 | 1556 | 0.103 | 1556 | 20% | 1816.0 |
| | | 0.6 | 1.403 | 1570 | 0.126 | 1570 | 0% | - |
| | 5 | 1.0 | 0.720 | 1676 | 0.120 | 1676 | 14% | 1816.0 |
| | | 0.8 | 1.608 | 1746 | 0.121 | 1746 | 0% | - |
| | | 0.6 | 1.966 | 1726 | 0.123 | 1726 | 0% | - |
| | 8 | 1.0 | 3.217 | 1584 | 0.156 | 1584 | 100% | 2143.3 |
| | | 0.8 | 13.730 | 1508 | 0.167 | 1508 | 100% | 2050.0 |
| | | 0.6 | 15.487 | 1560 | 0.159 | 1560 | 0% | - |
| 200 | 3 | 1.0 | 8.695 | 3306 | 0.140 | 3306 | 0% | - |
| | | 0.8 | 11.373 | 3112 | 0.155 | 3112 | 0% | - |
| | | 0.6 | 16.411 | 3190 | 0.174 | 3190 | 0% | - |
| | 5 | 1.0 | 3.172 | 3286 | 0.180 | 3286 | 0% | - |
| | | 0.8 | 1.395 | 3370 | 0.178 | 3370 | 0% | - |
| | | 0.6 | 1.451 | 3186 | 0.174 | 3186 | 0% | - |
| | 8 | 1.0 | 42.799 | 3118 | 0.243 | 3118 | 0% | - |
| | | 0.8 | 31.398 | 3260 | 0.261 | 3260 | 0% | - |
| | | 0.6 | 56.784 | 3334 | 0.244 | 3334 | 0% | - |

Table 4.1: JAP results

| \|J\| | \|T\| | div. | JAP rand. target | JAP target | improvement |
|---|---|---|---|---|---|
| 50 | 3 | 1.0 | 996.8 | 852 | 15% |
| | | 0.8 | 978.0 | 838 | 14% |
| | | 0.6 | 967.6 | 716 | 26% |
| | 5 | 1.0 | 1098.3 | 896 | 18% |
| | | 0.8 | 1047.3 | 854 | 18% |
| | | 0.6 | 896.0 | 826 | 8% |
| | 8 | 1.0 | 1292.1 | 846 | 35% |
| | | 0.8 | 1283.3 | 836 | 35% |
| | | 0.6 | 1353.8 | 820 | 39% |
| 100 | 3 | 1.0 | 2040.3 | 1744 | 15% |
| | | 0.8 | 1816.0 | 1556 | 14% |
| | | 0.6 | - | 1570 | - |
| | 5 | 1.0 | 1816.0 | 1676 | 8% |
| | | 0.8 | - | 1746 | - |
| | | 0.6 | - | 1726 | - |
| | 8 | 1.0 | 2143.3 | 1584 | 26% |
| | | 0.8 | 2050.0 | 1508 | 26% |
| | | 0.6 | - | 1560 | - |

Table 4.2: Improvement of JAP compared to randomized results

diversity decreases. This is due to the increasing sizes of the overlapping sets. In the most cases the runtime for 3 machine types is similar to the runtime for 5 machine types. However, for 8 machine types the runtime is much higher in most cases.

The optimal and randomized results are compared in Table 4.2. Only the instances with $|J| \in \{50, 100\}$ are considered since for $|J| = 200$ the randomized algorithm never finds a solution. For $|J| = 50$ the optimal value returned by JAPlin$_{\mathcal{I}}$ is in average 23% smaller than the randomized result. For $|J| = 100$ the optimal solution is about 18% smaller but it also has to be taken into account that the randomized algorithm finds only feasible solutions for 5 of the 9 instances.
Also some further instances, where the durations have a high variance and are allowed to be large compared to $f$, were generated. For these instances the

heuristic does not necessarily find the optimal solution value. Nevertheless, in practical applications the durations usually are of similar length since the jobs come from a common field. Further, the durations are much smaller than the production limit since the limit would not make sense, if it were already exceeded by a single job. Therefore, these unrealistic bad case instances are not presented here.

As already mentioned there are four methods to solve MJAP. The first is to solve MJAPlin$_{\mathcal{I}}$, the second to solve CMJAPlin$_{\mathcal{I}}$ and then CMJAP$_{x,s,z,t}$lin$_{\mathcal{I}}$, the third to solve MinOSlin$_{\mathcal{I}}$ and then JAPlin$_{\mathcal{I}}$ and the fourth to solve MinOSlin$_{\mathcal{I}}$, then CJAPlin$_{\mathcal{I}}$ and then CJAP$_{x,z,t}$lin$_{\mathcal{I}}$. In the implementations variables and constraints are created only if necessary, as already described previously. For MinOSlin$_{\mathcal{I}}$, for example, in this way around 97% of the variables and constraints can be saved. This reduces the size of the ILP and consequently the solving time and memory.
The results of the four methods are presented in Tables 4.3 and 4.4. Further, the results of a randomized algorithm are shown. Compared to JAP this randomized algorithm has also to randomly generate starting times for the movable jobs. The runtime for each run of the randomized algorithm is bounded by the time so solve MJAPlin$_{\mathcal{I}}$ (column MJAP $\rightarrow$ time). Like for JAP, the runtime of all algorithms increases with increasing $|K|$ and $|T|$ and decreasing diversity factor. In average the time to solve CMJAPlin$_{\mathcal{I}}$ is larger than the time to solve MJAPlin$_{\mathcal{I}}$. This is because in CMJAPlin$_{\mathcal{I}}$ much more binary variables are involved ($\mathcal{O}(|K||M| + |K|^2)$ versus $\mathcal{O}(|K|^2|T|)$). Therefore, solving MJAP via CMJAPlin$_{\mathcal{I}}$ and CMJAP$_{x,s,z,t}$lin$_{\mathcal{I}}$ is unfavorable, whereas solving MinOSlin$_{\mathcal{I}}$ and JAPlin$_{\mathcal{I}}$ is much faster. Even for $|K| = 200$, $|T| = 8$ and a diversity of 0.6 the runtime for this method does not exceed 12 seconds. However, the fastest method is solving MinOSlin$_{\mathcal{I}}$, CJAPlin$_{\mathcal{I}}$ and CJAP$_{x,z,t}$lin$_{\mathcal{I}}$. For all instances the runtime of this method is less than 0.405 seconds.

Like for JAP all four heuristics find the optimal solution for every single instance. The reason for this was already explained before. Nevertheless, instances can be specially designed for the MinOS method, for which the heuristic does not find the optimal solution. For example, if the fixed costs were set to 0 ($c_0 = 0$). MinOSlin$_{\mathcal{I}}$ minimizes the number of needed machines but with $c_0 = 0$ machines are for free. Thus, using more machines would surely improve the solution.

| \|K\| | \|T\| | div. | MJAP |  | CMJAP |  | MinOS+JAP |  |
|---|---|---|---|---|---|---|---|---|
|  |  |  | time | target | time | target | time | target |
| 50 | 3 | 1.0 | 0.356 | 780 | 0.504 | 780 | 0.321 | 780 |
|  |  | 0.8 | 0.250 | 698 | 0.186 | 698 | 0.584 | 698 |
|  |  | 0.6 | 0.396 | 636 | 1.280 | 636 | 0.814 | 636 |
|  | 5 | 1.0 | 0.473 | 796 | 1.957 | 796 | 0.295 | 796 |
|  |  | 0.8 | 0.459 | 758 | 2.018 | 758 | 0.210 | 758 |
|  |  | 0.6 | 0.517 | 686 | 15.112 | 686 | 0.331 | 686 |
|  | 8 | 1.0 | 0.703 | 752 | 1.229 | 752 | 0.360 | 752 |
|  |  | 0.8 | 0.435 | 742 | 2.280 | 742 | 0.549 | 742 |
|  |  | 0.6 | 1.011 | 728 | 26.175 | 728 | 0.548 | 728 |
| 100 | 3 | 1.0 | 0.476 | 1598 | 0.486 | 1598 | 2.317 | 1598 |
|  |  | 0.8 | 0.768 | 1296 | 1.728 | 1296 | 1.124 | 1296 |
|  |  | 0.6 | 1.125 | 1438 | 2.166 | 1438 | 1.566 | 1438 |
|  | 5 | 1.0 | 0.676 | 1396 | 3.292 | 1396 | 0.163 | 1396 |
|  |  | 0.8 | 0.792 | 1600 | 4.024 | 1600 | 0.394 | 1600 |
|  |  | 0.6 | 2.130 | 1582 | 32.210 | 1582 | 1.106 | 1582 |
|  | 8 | 1.0 | 3.512 | 1452 | 9.370 | 1452 | 3.599 | 1452 |
|  |  | 0.8 | 2.464 | 1256 | 29.232 | 1256 | 0.894 | 1256 |
|  |  | 0.6 | 6.018 | 1430 | 19.113 | 1430 | 3.051 | 1430 |
| 200 | 3 | 1.0 | 1.795 | 3030 | 2.469 | 3030 | 4.242 | 3030 |
|  |  | 0.8 | 1.455 | 2852 | 2.149 | 2852 | 9.761 | 2852 |
|  |  | 0.6 | 4.022 | 2924 | 4.622 | 2924 | 7.469 | 2924 |
|  | 5 | 1.0 | 1.369 | 3012 | 6.015 | 3012 | 0.782 | 3012 |
|  |  | 0.8 | 1.489 | 3088 | 16.853 | 3088 | 0.764 | 3088 |
|  |  | 0.6 | 2.281 | 2920 | 17.058 | 2920 | 0.823 | 2920 |
|  | 8 | 1.0 | 4.686 | 2598 | 58.185 | 2598 | 1.256 | 2598 |
|  |  | 0.8 | 49.850 | 2988 | 50.423 | 2988 | 11.959 | 2988 |
|  |  | 0.6 | 58.356 | 3056 | 45.978 | 3056 | 10.061 | 3056 |

Table 4.3: MJAP results (part 1)

| \|K\| | \|T\| | div. | MinOS+CJAP | | MJAP rand. | |
|---|---|---|---|---|---|---|
| | | | time | target | ratio | target |
| 50 | 3 | 1.0 | 0.115 | 780 | 97% | 1000.0 |
| | | 0.8 | 0.099 | 698 | 100% | 976.7 |
| | | 0.6 | 0.099 | 636 | 100% | 974.3 |
| | 5 | 1.0 | 0.107 | 796 | 95% | 1104.5 |
| | | 0.8 | 0.106 | 758 | 100% | 1046.5 |
| | | 0.6 | 0.137 | 686 | 22% | 900.0 |
| | 8 | 1.0 | 0.106 | 752 | 100% | 1331.9 |
| | | 0.8 | 0.120 | 742 | 100% | 1329.0 |
| | | 0.6 | 0.146 | 728 | 100% | 1326.0 |
| 100 | 3 | 1.0 | 0.151 | 1598 | 5% | 2057.0 |
| | | 0.8 | 0.135 | 1296 | 7% | 1816.0 |
| | | 0.6 | 0.156 | 1438 | 0% | - |
| | 5 | 1.0 | 0.165 | 1396 | 0% | - |
| | | 0.8 | 0.162 | 1600 | 0% | - |
| | | 0.6 | 0.175 | 1582 | 0% | - |
| | 8 | 1.0 | 0.175 | 1452 | 100% | 2149.7 |
| | | 0.8 | 0.182 | 1256 | 60% | 2064.3 |
| | | 0.6 | 0.200 | 1430 | 0% | - |
| 200 | 3 | 1.0 | 0.217 | 3030 | 0% | - |
| | | 0.8 | 0.300 | 2852 | 0% | - |
| | | 0.6 | 0.243 | 2924 | 0% | - |
| | 5 | 1.0 | 0.313 | 3012 | 0% | - |
| | | 0.8 | 0.270 | 3088 | 0% | - |
| | | 0.6 | 0.333 | 2920 | 0% | - |
| | 8 | 1.0 | 0.317 | 2598 | 0% | - |
| | | 0.8 | 0.405 | 2988 | 0% | - |
| | | 0.6 | 0.365 | 3056 | 0% | - |

Table 4.4: MJAP results (part 2)

| \|K\| | \|T\| | div. | MJAP rand. target | MJAP target | improvement |
|---|---|---|---|---|---|
| 50 | 3 | 1.0 | 1000.0 | 780 | 22% |
| | | 0.8 | 976.7 | 698 | 29% |
| | | 0.6 | 974.3 | 636 | 35% |
| | 5 | 1.0 | 1104.5 | 796 | 28% |
| | | 0.8 | 1046.5 | 758 | 28% |
| | | 0.6 | 900.0 | 686 | 24% |
| | 8 | 1.0 | 1331.9 | 752 | 44% |
| | | 0.8 | 1329.0 | 742 | 44% |
| | | 0.6 | 1326.0 | 728 | 45% |
| 100 | 3 | 1.0 | 2057.0 | 1598 | 22% |
| | | 0.8 | 1816.0 | 1296 | 29% |
| | | 0.6 | - | 1438 | - |
| | 5 | 1.0 | - | 1396 | - |
| | | 0.8 | - | 1600 | - |
| | | 0.6 | - | 1582 | - |
| | 8 | 1.0 | 2149.7 | 1452 | 32% |
| | | 0.8 | 2064.3 | 1256 | 39% |
| | | 0.6 | - | 1430 | - |

Table 4.5: Improvement of MJAP compared to randomized results

However, due to the practical irrelevance such instances are not considered in this analysis.

The optimal and randomized results are compared in Table 4.5. Only instances with $|K| \in \{50, 100\}$ are considered since for larger instances the randomized algorithm cannot find a solution in the given time. For $|K| = 50$ the optimal solution is in average 33% smaller than the randomized result. For $|K| = 100$ the optimal solution is in average 31% smaller than the randomized result. Note that the randomized algorithm finds only feasible solutions for 4 of the 9 instances.

In Table 4.6 the optimal solutions of the JAP instances are compared to the solutions of the corresponding MJAP instances, where jobs are allowed to be moved. The possible movement lies in a similar range as the duration of the jobs ($s^{\max} - s^{\min} \in \{0, 1, \ldots, 10\}$ and $d_k \in \{1, 2, \ldots, 10\}$). Thus, the average

| \|J\|/\|K\| | \|T\| | div. | JAP target | MJAP target | improvement |
|---|---|---|---|---|---|
| 50 | 3 | 1.0 | 852 | 780 | 8% |
| | | 0.8 | 838 | 698 | 17% |
| | | 0.6 | 716 | 636 | 11% |
| | 5 | 1.0 | 896 | 796 | 11% |
| | | 0.8 | 854 | 758 | 11% |
| | | 0.6 | 826 | 686 | 17% |
| | 8 | 1.0 | 846 | 752 | 11% |
| | | 0.8 | 836 | 742 | 11% |
| | | 0.6 | 820 | 728 | 11% |
| 100 | 3 | 1.0 | 1744 | 1598 | 8% |
| | | 0.8 | 1556 | 1296 | 17% |
| | | 0.6 | 1570 | 1438 | 8% |
| | 5 | 1.0 | 1676 | 1396 | 17% |
| | | 0.8 | 1746 | 1600 | 8% |
| | | 0.6 | 1726 | 1582 | 8% |
| | 8 | 1.0 | 1584 | 1452 | 8% |
| | | 0.8 | 1508 | 1256 | 17% |
| | | 0.6 | 1560 | 1430 | 8% |
| 200 | 3 | 1.0 | 3306 | 3030 | 8% |
| | | 0.8 | 3112 | 2852 | 8% |
| | | 0.6 | 3190 | 2924 | 8% |
| | 5 | 1.0 | 3286 | 3012 | 8% |
| | | 0.8 | 3370 | 3088 | 8% |
| | | 0.6 | 3186 | 2920 | 8% |
| | 8 | 1.0 | 3118 | 2598 | 17% |
| | | 0.8 | 3260 | 2988 | 8% |
| | | 0.6 | 3334 | 3056 | 8% |

Table 4.6: Improvement of MJAP compared to JAP

possible movement is about the average job duration. In practical applications jobs often can be moved in a wider range. However, as it is shown in Table 4.6, even in the case where jobs are allowed to be moved as much as their duration, an average improvement of already 11% can be achieved.
This improvement is independent of the size of $|J|/|K|$, $|T|$ and the diversity since the jobs are generated randomly such that they are distributed equally within a certain domain. Thus, all parts of the domain possess the same structure and the number of jobs or machine types have no impact on the improvement. Further, the diversity has no impact since the number of machines which can be saved by moving jobs is independent of the size of the overlapping sets. Larger overlapping sets do not necessarily imply that more machines can be saved. Nevertheless, for a larger range for $s^{\max} - s^{\min}$ even better results are expected.

# Chapter 5

# Conclusion

In this work two new classes of partition problems were introduced. The goal was to examine their complexity and to create efficient solving techniques. For partition problems with nonconstant weight functions mainly the complexity was studied, whereas for partition problems with convex target function the focus was on modeling of ILPs without discretizing the time. Further, either approximation algorithms or heuristics were developed.

Two different types of partition problems with nonconstant weight function were covered: Decision type problems and minimization type problems.
The decision type problems are $\mathbb{NP}$-complete even if the weight function possesses certain monotonic properties. Thus, a heuristic based on integer linear programming was developed. This heuristic iteratively creates partitions, where the last partition is used to estimate the weights of the next partition. This is done until a feasible partition is found. The complexity of the problem can also be seen in the numerical results. For many instances for which a heuristic finds a solution in a few seconds the expected chance of a randomized approach to find a solution is zero.
For the minimization type problems the situation is quite different. For decision type problems the hardness is independent of the monotonic properties of the weight function, whereas for minimization type problems these properties have an impact on the hardness. For the general case a simple heuristic was developed. The idea is to start with some random partition and then iteratively move or switch elements between the partitioning sets such that the target function decreases. For instances in which the monotonic properties hold fast approximation algorithms were devised. To get even better solutions the approximation results were used as starting partitions for the heuristic. As for the decision type problems, the developed solution techniques for the minimization type problems

perform significantly better than a randomized approach.
In summary, for both problem types fast algorithms were designed that produce really good solutions compared to a random approach.

Partition problems with convex target function were specified as special job assignment problems. For these problems two different versions with fixed and variable starting times were introduced. To derive ILPs for both problems logical dependencies between descriptive variables were established. The according linear inequalities were generated and the correctness was verified. Deliberately an approach that does not need discretization of the time was chosen. In this way even long periods can be tackled. The ILPs produce optimal solutions. To solve larger problem instances heuristics were developed afterwards. These are also implemented as ILPs. For variable starting times a preprocessing heuristic which arranges the jobs such that as few as possible machines are needed was presented. For this heuristic an extensive analysis of the feasible region was given to single out the cases where variables and constraints can be saved in the formulation. Further, for fixed as well as for variable starting times heuristics which first assign jobs to machine types and afterwards to single machines were devised. Finally, the heuristics were combined. This approach results in two ways to solve the fixed starting times version and four ways to solve the variable starting times version.
The numerical results are really convincing. For large instances, which still can be solved optimally in less than 1 minute, randomized approaches cannot find a feasible solution at all in comparable time. Further, if a randomized algorithm finds a solution it is in average 20% to 30% worse than the optimal solution. This also underlines the complexity of the partition problems. Another very interesting fact is that the heuristics even find the optimal solution for all instances and this in less than a second. Only for artificial designed instances, which are assumed to be irrelevant for practical applications, the heuristics fail.

In summary it can be said that an interesting set of new problem definitions, which are motivated by their occurrence in practical applications, was proposed in this work. Further, exact algorithms, approximation algorithms and heuristics were developed, which solve the problems in phenomenal time. Due to the fact that all problems were firstly introduced in this work no other solving techniques yet exist. Thus, all algorithms are compared with randomized algorithms. Especially the heuristics often run in milliseconds and deliver outstanding solutions, which are up to 56% better than the randomized result.

A lot of future research is imaginable in the context with the presented partition problems. A few thoughts and considerations are given in the following.

In Chapter 3 subset functions were defined. These functions weight single elements of $V$ depending on a subset of $V$. However, it would also be interesting to weight tuples of elements in a nonconstant way. Thus, problems like CSC could be formulated such that distances between points are variable. This would increase the area of application for the problem. Further, additional properties, like some sort of convexity or subadditivity, could be introduced for subset and cumulated subset functions. It would be interesting to see how the complexity of FPP, MinSumPP and MinMaxPP changes if the weight function has one or multiple of these properties.
For FPP a heuristic was developed, which often finds a feasible solution. However, there is no guarantee for that. To create an algorithm which finds a feasible solution if one exists would be of great advantage. Also criteria which can identify feasible or infeasible instances of FPP would be a considerable benefit.
Moreover, two minimization type problems, MinSumPP and MinMaxPP, were studied. However, also many other target functions could be analyzed. For example for each subset some target value could be introduced and the deviation from this value is punished with a penalty factor. In this setting the goal would be to minimize the total penalty. Further, approximation algorithms with better approximation ratios should be developed. Especially for MinSumPP this is important since the approximation is sometimes worse than the expected randomized result. Constant approximation ratios which do not depend on $V$ or $k$ would be of great value. Better approximations would also lead to better solutions for the heuristics if these are used as starting partitions.

For JAP and MJAP the focus was mainly on the modeling of ILPs. Already in some cases criteria were provided which describe the necessity of variables and constraints. The search for such criteria should be extended to all developed programs. Further, it would be helpful to find a smaller ILP representation for all problems. Less variables and constraints often entail a shorter runtime and less memory usage.
In this work piecewise linear convex functions with two different gradients were

considered. The models can easily be adapted to functions with more different gradients. Further, in general convex functions can be approximated by piecewise linear convex functions. Thus, JAP and MJAP could be introduced with general convex functions and solved with the existing methods, where the convex function is approximated by a piecewise linear convex function. In this context the question would be how much this approximation affects the solution value. The cost vector $c$ was defined without any restrictions. It would be interesting to see whether the complexity changes if $c$ is restricted in some way. For example, if $c_2 - c_1$ is bounded by some value or if $c_0$ must be larger than a certain value. The role of $f$ could also be analyzed in further studies.
For JAP and MJAP exact algorithms or heuristics were introduced. Especially the heuristics have a very short runtime and produce excellent solutions. However, approximation algorithms, which run in polynomial time and guarantee a certain solution quality, would be of great advantage.

Altogether, it can be summarized that the special partition problems which were introduced in this work open up a whole area for further research. Advancing this research will make it possible to express problems in a more general context and thus to reach even more practical applications.

# Bibliography

[1] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *The 8th IEEE international conference on advanced computing and communications (ADCOM 2000)*, pages 45–52, 2000.

[2] D. Alejo, J. M. Díaz-Báñez, J. A. Cobano, P. Pérez-Lantero, and A. Ollero. The velocity assignment problem for conflict resolution with multiple aerial vehicles sharing airspace. *Journal of Intelligent & Robotic Systems*, 69(1-4):331–346, 2013.

[3] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

[4] N. Azi, M. Gendreau, and J.-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763, 2010.

[5] R. Baldacci, A. Mingozzi, R. Roberti, and R. W. Calvo. An exact algorithm for the two-echelon capacitated vehicle routing problem. *Operations Research*, 61(2):298–314, 2013.

[6] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. Naor, and R. Schwartz. Min-max graph partitioning and small set expansion. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 17–26. IEEE, 2011.

[7] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of operations Research*, 16(1):199–240, 1988.

[8] S. Borgwardt, A. Brieden, and P. Gritzmann. Constrained minimum-$k$-star clustering and its application to the consolidation of farmland. *Operational Research*, 11(1):1–17, 2011.

[9] A. Brieden, P. Gritzmann, and B. Reinbold. A partition problem with convex target function. in preparation, 2014.

[10] A. Brieden and B. Reinbold. Nonconstant weight partition problems. in preparation, 2014.

[11] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[12] Rainer E Burkard and Tilman Bönniger. A heuristic for quadratic boolean programs with applications to quadratic assignment problems. *European Journal of Operational Research*, 13(4):374–386, 1983.

[13] K. Cameron, E. M. Eschen, C. T. Hoàng, and R. Sritharan. The list partition problem for graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–399. Society for Industrial and Applied Mathematics, 2004.

[14] S. Ceschia, L. Di Gaspero, and A. Schaerf. Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs. *Journal of Scheduling*, 14(6):601–615, 2011.

[15] R.-S. Chang, J.-S. Chang, and P.-S. Lin. An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems*, 25(1):20–27, 2009.

[16] T. C. E. Cheng, W.-C. Lee, and C.-C. Wu. Scheduling problems with deteriorating jobs and learning effects including proportional setup times. *Computers & Industrial Engineering*, 58(2):326–331, 2010.

[17] T. C. E. Cheng, S.-J. Yang, and D.-L. Yang. Common due-window assignment and scheduling of linear time-dependent deteriorating jobs and a deteriorating maintenance activity. *International Journal of Production Economics*, 135(1):154–161, 2012.

[18] D. M.-H. Chiang, C.-P. Lin, and M.-C. Chen. The adaptive approach for storage assignment by mining data of warehouse management system for distribution centres. *Enterprise Information Systems*, 5(2):219–234, 2011.

[19] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Transactions on Algorithms (TALG)*, 1(2):350–366, 2005.

[20] B. J. Conijn and W. P. M. Nuijten. Modeling and solving the vehicle routing problem with stochastic travel times and hard time windows. *Eindhoven University of Technology, Department of Mathematics and Computer Science*, 2013.

[21] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. Vrp with time windows. *The vehicle routing problem*, 9:157–193, 2002.

[22] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[23] G. B. Dantzig. *Linear programming and extensions.* Princeton university press, 1963.

[24] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[25] Reinhard Diestel. *Graphentheorie.* Springer, 4. edition, 2006.

[26] G. A. Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.

[27] R. Driessel and L. Mönch. Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. *Computers & Industrial Engineering*, 61(2):336–345, 2011.

[28] M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with split deliveries. *Discrete Applied Mathematics*, 50(3):239–254, 1994.

[29] A. Elmi, M. Solimanpur, S. Topaloglu, and A. Elmi. A simulated annealing algorithm for the job shop cell scheduling problem with

intercellular moves and reentrant parts. *Computers & Industrial Engineering*, 61(1):171–178, 2011.

[30] M. Elwekeil, M. Alghoniemy, H. Furukawa, and O. Muta. Lagrangian relaxation approach for low complexity channel assignment in multi-cell wlans. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 138–142. IEEE, 2013.

[31] T. Feo, O. Goldschmidt, and M. Khellaf. One-half approximation algorithms for the k-partition problem. *Operations Research*, 40(1-Supplement-1):S170–S173, 1992.

[32] FICO. `http://www.fico.com/en/Pages/default.aspx`, 2013. [Online; accessed 01.10.2013].

[33] FICO. IVE Editor. `http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-IVE.aspx`, 2013. [Online; accessed 01.10.2013].

[34] FICO. Xpress-Mosel. `http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Mosel.aspx`, 2013. [Online; accessed 01.10.2013].

[35] M. A. Figliozzi. An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows. *Transportation Research Part C: Emerging Technologies*, 18(5):668–679, 2010.

[36] B. Fu, H. Jiang, B. Yang, and B. Zhu. Exponential and polynomial-time algorithms for the minimum common string partition problem. In *Combinatorial Optimization and Applications*, pages 299–310. Springer, 2011.

[37] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.

[38] S. I. Gass. *Linear programming: Methods and Applications.* McGraw-Hill Book Company, 1985.

[39] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial & Applied Mathematics*, 10(2):305–313, 1962.

[40] B. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

[41] I. Gribkovskaia, Ø. Halskau, and K. N. B. Myklebost. Models for pick-up and deliveries from depots with lasso solutions. *NOFOMA2001, Collaboration in logistics: connecting islands using information technology*, pages 279–293, 2001.

[42] P. Gritzmann. *Grundlagen der Mathematischen Optimierung*. Springer, 2013.

[43] G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte, and V. Marianov. A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research*, 206(2):341–349, 2010.

[44] N. G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525, 1996.

[45] C. A. Hjorring. *The vehicle routing problem and local search metaheuristics*. PhD thesis, University of Auckland, 1995.

[46] I. Holyer. The np-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

[47] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*, volume 2. Pearson Studium, 2. edition, 2002.

[48] J. Hromkovič. *Theoretische Informatik*. Springer, 2010.

[49] X. Huang, J.-B. Wang, L.-Y. Wang, W.-J. Gao, and X.-R. Wang. Single machine scheduling with time-dependent deterioration and exponential learning effect. *Computers & Industrial Engineering*, 58(1):58–63, 2010.

[50] K. Jansen and M. Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit.* Walter de Gruyter, 2008.

[51] M. Ji and T. C. E. Cheng. Scheduling with job-dependent learning effects and multiple rate-modifying activities. *Information Processing Letters*, 110(11):460–463, 2010.

[52] R. M. Karp. Reducibility among combinatorial problems. *50 Years of Integer Programming 1958-2008*, 2010.

[53] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

[54] B. H. Korte and J. Vygen. *Kombinatorische Optimierung: Theorie und Algorithmen.* Springer, 2012.

[55] L. G. Kroon, A. Sen, H. Deng, and A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 279–292. Springer, 1997.

[56] E. L. Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963.

[57] C. Y. Lee and S. J. Kim. Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights. *Computers & industrial engineering*, 28(2):231–243, 1995.

[58] D. Lei. Fuzzy job shop scheduling problem with availability constraints. *Computers & Industrial Engineering*, 58(4):610–617, 2010.

[59] J. K. Lenstra and A. H. G. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[60] J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[61] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[62] J.-q. Li and Q.-k. Pan. Chemical-reaction optimization for solving fuzzy job-shop scheduling problem with flexible maintenance activities. *International Journal of Production Economics*, 2012.

[63] Y. Li, B. Wang, and L. A. Caudillo-Fuentes. Modeling a hotel room assignment problem. *Journal of Revenue & Pricing Management*, 12(2):120–127, 2012.

[64] S. Lorini, J.-Y. Potvin, and N. Zufferey. Online vehicle routing and scheduling with dynamic travel times. *Computers & Operations Research*, 38(7):1086–1090, 2011.

[65] D. Mahjoub, A. Leskovskaya, and D. W. Matula. Approximating the independent domatic partition problem in random geometric graphs-an experimental study. In *CCCG*, pages 195–198, 2010.

[66] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *Knowledge and Data Engineering, IEEE Transactions on*, 11(5):769–778, 1999.

[67] J. E. Mendoza, B. Castanier, C. Guéret, A. L. Medaglia, and N. Velasco. A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. *Computers & Operations Research*, 37(11):1886–1898, 2010.

[68] M. Mika, G. Waligora, and J. Węglarz. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. *European Journal of Operational Research*, 187(3):1238–1250, 2008.

[69] E. Moradi, S. M. T. Fatemi Ghomi, and M. Zandieh. Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem. *Expert systems with applications*, 38(6):7169–7178, 2011.

[70] G. Mosheiov. A note: Multi-machine scheduling with general position-based deterioration to minimize total load. *International Journal of Production Economics*, 135(1):523–525, 2012.

[71] S. U. Ngueveu, C. Prins, and R. W. Calvo. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 37(11):1877–1885, 2010.

[72] J. Oppen, A. Løkketangen, and J. Desrosiers. Solving a rich vehicle routing and inventory problem using column generation. *Computers & Operations Research*, 37(7):1308–1317, 2010.

[73] C. S. Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[74] I. A. Pirwani and M. R. Salavatipour. A weakly robust ptas for minimum clique partition in unit disk graphs. In *Algorithm Theory-SWAT 2010*, pages 188–199. Springer, 2010.

[75] T. Ralphs, J. Hartman, and M. Galati. Capacitated vehicle routing and some related problems. *Rutgers University*, 2001.

[76] M.-E. Rancourt, J.-F. Cordeau, and G. Laporte. Long-haul vehicle routing and scheduling with working hour rules. *Transportation Science*, 47(1):81–107, 2013.

[77] L. B. Reinhardt, M. K. Jepsen, and D. Pisinger. The vehicle routing problem with edge set costs. *Branch-and-cut and Branch-and-Cut-and-Price Algorithms for Solving Vehicle Routing Problems*, page 179, 2011.

[78] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[79] H. M. Salkin. *Integer programming.* Addison-Wesley Reading, 1975.

[80] A. Schrijver. *Theory of linear and integer programming.* Wiley, 1998.

[81] C. J. Schuster and J. M. Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31(4):308–318, 2003.

[82] D. Solow. *Linear programming: An introduction to finite improvement algorithms.* North-Holland New York, 1984.

[83] S. Song, K. Hwang, and Y.-K. Kwok. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *Computers, IEEE Transactions on*, 55(6):703–719, 2006.

[84] A. Soukhal, A. Oulamara, and P. Martineau. Complexity of flow shop scheduling problems with transportation constraints. *European Journal of Operational Research*, 161(1):32–41, 2005.

[85] D. M. Tate and A. E. Smith. A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 22(1):73–83, 1995.

[86] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[87] D. Wang, M.-Z. Wang, and J.-B. Wang. Single-machine scheduling with learning effect and resource-dependent processing times. *Computers & Industrial Engineering*, 59(3):458–462, 2010.

[88] R. Wanka. *Approximationsalgorithmen*. Springer, 2006.

[89] J.-H. Yan and G. J. Chang. The path-partition problem in block graphs. *Information processing letters*, 52(6):317–322, 1994.

[90] S.-J. Yang, D.-L. Yang, and T. C. E. Cheng. Single-machine due-window assignment and scheduling with job-dependent aging effects and deteriorating maintenance. *Computers & Operations Research*, 37(8):1510–1514, 2010.