Model-based approaches for large-scale optimization in business operations

INAUGURALDISSERTATION

zur Erlangung der Würde eines Doctor rerum oeconomicarum
der Wirtschafts- und Sozialwissenschaftlichen Fakultät der Universität Bern

Tamara Bigler

Erstbetreuer: Prof. Dr. Philipp Baumann
Professur für Quantitative Methoden der BWL
Departement Betriebswirtschaftslehre
Engehaldenstrasse 4, 3012 Bern

Bern, Januar 2023

Die Fakultät hat diese Arbeit am 30. März 2023 auf Antrag der beiden Gutachter Prof. Dr. Olivier Gallay und Prof. Dr. Philipp Baumann als Dissertation angenommen, ohne damit zu den darin ausgesprochenen Auffassungen Stellung nehmen zu wollen.

Contents

Introduction

Companies nowadays have to operate in an increasingly competitive and complex environment. Under these challenging conditions, it has become essential for them to optimize their business operations, i.e., the activities that they must conduct on a regular, often daily, basis. The nature of these business operations strongly varies between companies. For a pharmaceutical company, an important business operation is, for example, the scheduling of their research activities. With improved scheduling, new drugs are brought to markets earlier, which can lead to a decisive competitive advantage. For a telecommunications company, an important business operation is, for example, the promotion of new products and services to existing customers. Contacting the right customers for the right products may lead to an increase in sales and profitability of these products. Many business operations, including the two examples from above, can be improved by solving mathematical optimization problems with techniques from the field of Operations Research. An optimization problem consists of the decisions to be taken, the constraints that define the set of feasible decisions, and an objective that is either maximized (profit) or minimized (project duration). In the case of the telecommunications company, the decisions to be taken are which customers are contacted for which product on which day. An example of a constraint is an overall budget that cannot be exceeded, and an example of the objective is the maximization of the total expected profit that results from contacting the customers.

A standard approach for solving such an optimization problem is first to express the problem as a mathematical model and then use standard optimization software, known as a solver, to find the best possible solution. A great advantage of this approach is that the mathematical model can easily be adjusted to changes in the underlying problem. This flexibility is required in a dynamic business environment where constraints or objectives may change over time. However, a major drawback of this standard approach is its limited scalability when applied to specific types of complex optimization problems. For these problems, the generic solvers fail to find the best or even a good solution in a reasonable running time. Specialized algorithms, so-called heuristics, are required instead. Heuristics

apply problem-specific search strategies to derive a good solution to an optimization problem quickly. However, because these heuristics are designed for specific optimization problems, they are difficult to adapt if the constraints or the objective of the optimization problem change. A solution technique that has been shown to be both flexible and scalable for complex optimization problems are matheuristics. Matheuristics are model-based approaches that decompose an optimization problem into smaller subproblems and solve these subproblems using mathematical models. Essential for the performance of a matheuristic is how the problem is decomposed into subproblems, which is an important field of research in Operations Research.

This thesis contributes to this field of research by introducing model-based approaches for large-scale optimization in business operations. It consists of three papers on three specific optimization problems in direct marketing, project management, and facility location. Real-world instances of all three of these problems involve a large number of customers, activities, or facilities and require the flexibility to incorporate practical constraints easily. To address these challenges, we developed three matheuristics. The matheuristics employ innovative problem decomposition strategies and outperform state-of-the-art approaches on large-scale instances.

In the first paper, we study a customer assignment problem from a major telecommunications company. The telecommunications company runs different direct marketing campaigns to promote its products and services. The goal of the telecommunications company is to assign the customers to the direct marketing campaigns so that the total expected profit is maximized. Thereby, various business constraints, such as budgets and sales constraints, must be considered. Also, different customer-specific constraints ensure that each customer is not assigned to a direct marketing campaign too frequently. A particular challenge is the size of practical problem instances. These instances involve millions of customers and hundreds of direct marketing campaigns. The methodological contribution of this paper consists of decomposing the optimization problem into two subproblems that each can be solved efficiently. In the first subproblem, customers are assigned to campaigns based on their membership to a customer group. In the second subproblem, individual customers are assigned to campaigns based on the solution that was derived in the first subproblem. The unique feature of our decomposition strategy is that the customer-specific constraints are already considered in the first subproblem, even though the first subproblem deals with groups of customers and not individual customers. In an experimental analysis based on numerous generated and real-world instances, we can demonstrate that even though we decompose the problem, the resulting solutions are still of very high quality. The matheuristic has been deployed in the company and is now

used daily. In a proof of benefit conducted by the company based on a selected campaign, they observed that using the matheuristic increased the number of sales by 90%, resulting in an improvement in the profitability of this campaign by 300%.

The second paper deals with a project scheduling problem that often arises in the pharmaceutical industry, where research activities, e.g., clinical tests, can be executed at different locations, e.g., research labs. The problem consists of determining a start time for each activity, selecting a location for the execution of each activity, and assigning resource units, e.g., research staff or equipment, to the execution of the activities. Various practical constraints must be considered, such as transportation times that arise when, e.g., a resource unit must be transported from one location to another. With only a few activities involved, the number of possible schedules can already grow very large. We developed a mathematical model and, based on this model, a novel matheuristic for this problem. The main methodological contribution of the matheuristic is its problem decomposition strategy. Instead of dividing the project into subprojects, the model in the matheuristic is set up for all project activities. However, the solver makes some decisions only for a subset of the activities. To schedule an entire project, multiple iterations have to be performed, where in each iteration, another subset of activities is considered. This iterative decision process substantially reduces running times compared to when all decisions are conducted simultaneously. In a computational experiment, the novel model outperforms the leading model from the literature on small instances. The matheuristic outperforms the state-of-the-art heuristics on all considered performance metrics on larger instances.

In the third paper, we consider the problem of locating obnoxious facilities. Obnoxious means that the facilities negatively affect their nearby environment and should thus be located far away from clients. Examples of obnoxious facilities are waste plants, oil refineries, and wind turbines. The problem consists of opening from a set of potential locations a given number of facilities such that the open facilities are far away from the clients. We further study an extension of this problem that includes practical constraints which limit the number of facilities that can be opened in certain regions of an instance. Our matheuristic starts from an initial solution and iteratively improves the solution by removing and adding facilities. The quality of the final solution (after the improvement iterations) strongly depends on the initial solution. When two very similar initial solutions are provided, the likelihood of finding very similar final solutions is high. One main methodological contribution is a procedure that we designed, which is guaranteed to generate initial solutions that are very different from each other. This diversification in the initial solutions increases the likelihood of finding high-quality final solutions.

The matheuristic outperforms the state-of-the-art metaheuristics on instances including thousands of clients and potential locations for facilities.

Even though we consider three specific optimization problems in this thesis, the contributions of the three papers can be generalized and applied to related problems and thus advance the state of knowledge in the field of large-scale optimization.

Paper I

A matheuristic for a customer assignment problem in direct marketing [1]

Tamara Bigler Manuel Kammermann Philipp Baumann

Department of Business Administration
University of Bern

Contents

[1] Published in European Journal of Operational Research 304, 689–708 (DOI:10.1016/j.ejor.2022.04.009)

Abstract

In direct marketing, companies use sales campaigns to target their customers with personalized product offers. The effectiveness of direct marketing greatly depends on the assignment of customers to campaigns. In this paper, we consider a real-world planning problem of a major telecommunications company that assigns its customers to individual activities of its direct marketing campaigns. Various side constraints, such as budgets and sales targets, must be met. Conflict constraints ensure that individual customers are not assigned too frequently to similar activities. Related problems have been addressed in the literature; however, none of the existing approaches cover all the side constraints considered here. To close this gap, we develop a matheuristic that employs a new decomposition strategy to cope with the large number of conflict constraints in typical problem instances. In a computational experiment, we compare the performance of the proposed matheuristic to the performance of two mixed-binary linear programs on a test set that includes large-scale real-world instances. The matheuristic derives near-optimal solutions in short running times for small- to medium-sized instances and scales to instances of practical size comprising millions of customers and hundreds of activities. The deployment of the matheuristic at the company has considerably increased the overall effectiveness of its direct marketing campaigns.

1.1 Introduction

Companies in competitive sectors such as banking and telecommunications strongly rely on direct marketing to promote their products (Miguéis et al., 2017). In direct marketing campaigns, companies contact their customers individually via *call*, *direct mail*, *email*, or *text message* to make a personalized offer. The success of such campaigns greatly depends on the assignment of customers to campaigns. Increasing the overall response rate by targeting the right customers can result in a substantial profit increase. However, contacting individual customers too frequently and offering products they are not interested in, negatively impacts the effectiveness of direct marketing.

We introduce a real-world planning problem of a major telecommunications provider. The problem input consists of a set of activities and a set of customers. Each activity is scheduled for a specific day, and the eligible customers, i.e., the customers who can

be assigned, are known for each activity. An expected profit, a response probability, and a cost are given for every possible assignment of a customer to an activity. Various business and customer-specific constraints must be considered. The business constraints include assignment constraints that control the number of assignments to specific activities, budget constraints that ensure that the total cost associated with assignments to specific activities does not exceed a prescribed budget, as well as sales constraints that control the expected number of sales resulting from assignments to specific activities. The customer-specific constraints include lower and upper bounds on the number of contacts per customer within a prespecified time window (e.g., one month) and conflict constraints that ensure that each customer who is eligible for two conflicting activities is assigned to at most one of the two activities. Two activities are in conflict when they take place in close succession and are associated with the same channel or promote the same product. The objective is to assign the customers to the activities such that the total expected profit is maximized. The company solves this planning problem on a daily basis, each time with updated data. Before using our approach, a dedicated team of the company used to manually assign its customers to the activities. To preserve the existing workflow, a key requirement for the solution approach was that it is capable of producing solutions for practical instances within 30 minutes.

To the best of our knowledge, none of the existing approaches from the literature can be directly applied to the above-described planning problem. Approaches that have been proposed for optimizing direct marketing campaigns only consider subsets of the constraints of our problem setting (cf. Table 1.5 in Section 1.3.1 for an overview). In particular, none of the approaches consider customer-specific conflict constraints. Conflict constraints in a more general sense have received considerable attention in the literature on related planning problems such as the assignment problem and the bin packing problem. However, approaches for these problems also cannot be applied directly to our problem because they do not allow the assignment of the same customer to multiple activities or the assignment of multiple customers to the same activity. The planning problem here also differs from related planning problems in terms of the size of typical instances. Practical instances involve millions of customers and hundreds of activities, which may lead to hundreds of millions of conflict constraints. This large number of conflict constraints hinders the use of exact approaches that employ a mixed-binary linear programming formulation of the entire planning problem. With hundreds of millions of conflict constraints, the time required to construct the model alone exceeds the running time limit prescribed by the company. The large number also makes it difficult to adapt existing heuristics that were not specifically designed to address this challenge. Hence,

the development of a new solution approach is required.

We propose a matheuristic for the above-described problem. The matheuristic follows the idea of solving a mathematical model for groups of customers rather than individual customers. The main methodological feature of the matheuristic is the problem decomposition strategy, which allows grouping of customers while still enforcing conflict constraints for individual customers. The decomposition works in four steps. In the first step, customers who are eligible for the same activities are grouped together such that the customers in one group are subject to the same conflicts among activities. In the second step, a clustering algorithm further divides each group into subgroups such that the customers in the same subgroup have similar expected profits. In the third step, a linear program decides how many customers of a subgroup are assigned to an activity. To consider conflicts among activities, the linear program employs new types of constraints which are defined for sets of activities of maximal size in which every two distinct activities are conflicting. To derive these sets efficiently without introducing redundancies, we developed a preprocessing technique. In the fourth step, an iterative algorithm assigns individual customers to the activities in a carefully selected sequence based on the solution derived by the linear program. The trade-off between solution quality and running time can be controlled by changing the number of subgroups created in the second step. The proposed decomposition scheme is the first to show how conflict constraints can be grouped and effectively incorporated into an aggregated optimization model. We believe that these ideas can be useful for the development of heuristics for related combinatorial optimization problems with conflict constraints.

In an experimental analysis, we use 27 generated and 13 real-world instances to compare the performance of the matheuristic to the performance of two mixed-binary linear programming formulations. The matheuristic finds near-optimal solutions in short running times for instances that could be solved to optimality by at least one of the two mixed-binary linear programming formulations. For all other instances, the matheuristic clearly outperforms both mixed-binary linear programming formulations in terms of solution quality and running time. We find that the preprocessing technique and the new types of constraints in the linear program contribute substantially to the effectiveness and the speed of the matheuristic. The matheuristic has been successfully deployed at the company and is now used on a daily basis. According to the company, introducing the matheuristic led to a substantial increase in the overall profitability of the campaigns. Moreover, the problem decomposition strategy of the matheuristic allows the company to approximate the impact of strategic decisions (e.g., an increase of budgets) in near real time by solving the linear program of the third step several times with different

parameters.

The rest of the paper is structured as follows. In Section 1.2, we describe the planning problem in more detail. In Section 1.3, we review the related literature. In Section 1.4, we formulate the planning problem as a mixed-binary linear program. In Section 1.5, we describe the four steps of the matheuristic. In Section 1.6, we explain the preprocessing technique used in the matheuristic in more detail and introduce an alternative mixed-binary linear programming formulation that makes use of the preprocessing technique. In Section 1.7, we report the results. Finally, in Section 1.8, we draw conclusions and give directions for future research.

1.2 Planning problem

In Section 1.2.1, we provide the business context of the planning problem. In Section 1.2.2, we specify the planning problem. In Section 1.2.3, we illustrate the planning problem with an example.

1.2.1 Business context

The telecommunications company that reported the planning problem simultaneously runs multiple direct marketing campaigns to promote its products and services related to different market segments to existing customers of the company. Each campaign is created by a marketing manager who selects a set of target products, designs the offer, identifies eligible customers, and determines the activities of the campaign, i.e., the days on which customers can be contacted via specific channels. The marketing managers also define some of the business constraints such as assignment constraints for activities of their campaigns. Other business constraints such as budgets are defined in a centralized manner by higher management. The assignment of the company's customers to its activities is performed by a central unit instead of the campaign managers to prevent customers who are eligible for activities of multiple campaigns from being contacted too frequently. The central unit considers all business constraints when assigning the customers to the activities. To replace the current practice of manually assigning the customers to the activities, the company asked us to develop a heuristic to assign its customers automatically. In the next section, we describe the planning problem from the perspective of the central unit.

1.2.2 Problem description

The problem input consists of a set of activities and a set of customers. Each customer can be assigned to one or multiple activities. An expected profit is given for each possible assignment. The goal is to assign the customers to the activities such that the total expected profit is maximized subject to various constraints. We distinguish between business and customer-specific constraints. The business constraints consist of:

- **Minimum assignment constraints**, which impose lower bounds on the number of assignments to sets of selected activities, and **maximum assignment constraints**, which impose upper bounds on the number of assignments to sets of selected activities. These constraints balance the number of assignments over activities and can be used, for example, to control the utilization of specific channels.

- **Budget constraints**, which impose upper bounds on the total costs that result from assignments to sets of selected activities. The cost per assignment depends on the channel over which a customer is contacted, and hence may differ among activities. These constraints ensure that funds allocated to individual channels or groups of channels are not exceeded.

- **Minimum sales constraints**, which impose lower bounds on the number of expected sales that result from assignments to sets of selected activities, and **maximum sales constraints**, which impose upper bounds on the number of expected sales that result from assignments to sets of selected activities. Each possible assignment is associated with a response probability, which states the likelihood of a positive customer reaction (i.e., a sale of the target product). The expected sales that result from assignments to a set of selected activities are computed as the sum of the corresponding response probabilities. These constraints enable the company to control the intensity of promoting new products or services. Maximum sales constraints are useful to prevent an overload of sales channels. For example, they can be used to distribute the customer volume in shops over time.

Each activity is associated with different characteristics, such as the day on which the assigned customers are contacted, the channel that is used to contact the customers, and the target products that are promoted. These characteristics allow definition of business constraints for specific time periods, channels, or target products by selecting the corresponding activities. More complicated selections of activities based on combinations of these characteristics are also possible. For example, minimum and maximum assignment

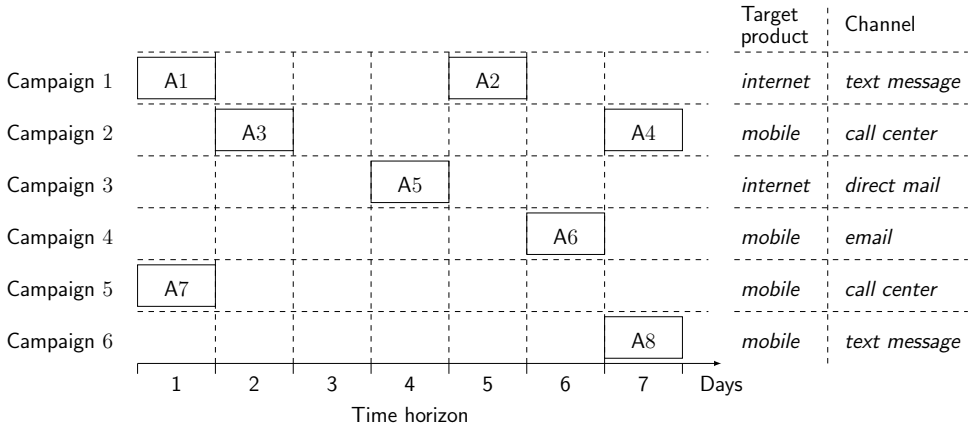

Figure 1.1: Activities (A1 to A8) of the illustrative example

constraints are typically defined for specific combinations of channels and time periods. The customer-specific constraints consist of:

- **Minimum contact constraints**, which impose a lower bound on the number of times that a customer is assigned to a set of selected activities, and **maximum contact constraints**, which impose an upper bound on the number of times that a customer is assigned to a set of selected activities. These constraints ensure that a customer is not contacted too frequently. The contact constraints are derived from contact rules, which may state, for example, that a customer cannot be contacted more than twice in January.

- **Conflict constraints**, which ensure that each customer is assigned to at most one out of two conflicting activities. A conflict arises when two activities take place within a certain number of consecutive days and use certain combinations of channels and target products. These constraints ensure that a customer is not assigned too frequently to similar activities. The conflict constraints are derived from conflict rules, which may state, for example, that a customer cannot be contacted twice via a *call* within one week. A separate conflict constraint would be imposed for each customer and each pair of activities which use the channel *call center* and take place within seven consecutive days.

This planning problem is strongly NP-hard, as it can be reduced to a generalized assignment problem that is known to be NP-hard (cf. Garey and Johnson, 2002). The minimum assignment constraints, the minimum and maximum sales constraints, and the minimum contact constraints reflect strategic decisions rather than operational requirements. Also, some of these constraints are defined independently by different marketing

Table 1.1: Activities of the illustrative example

Activity	Day	Channel	Target products	Cost
A1	1	*text message*	*internet*	1
A2	5	*text message*	*internet*	1
A3	2	*call center*	*mobile*	10
A4	7	*call center*	*mobile*	10
A5	4	*direct mail*	*internet*	4
A6	6	*email*	*mobile*	1
A7	1	*call center*	*mobile*	10
A8	7	*text message*	*mobile*	1

managers. This process may lead to contradictory constraints, for example, it is possible that two marketing managers each define a minimum assignment constraint that individually could be satisfied but together cannot be satisfied because of budget or conflict constraints. These dependencies get more complex if all types of minimum constraints are considered. Thus, the company wants to treat the constraints of the aforementioned four constraint types (minimum assignment, minimum and maximum sales, and minimum contact constraints) in almost all instances as soft constraints that can be violated subject to a penalty. The rest of the constraints must be treated as hard constraints because they represent operational requirements. The main purpose of using the soft constraints is to be able to generate a solution even if some constraints cannot be satisfied. Another advantage of the soft constraints is that users at the company are able to observe which slack variables take a positive value and thus, which constraints cannot be satisfied in the current setting. The penalty is computed for every soft constraint by multiplying the difference between the achieved assignments (or sales) and the prescribed bound by a constant. The constant can be set individually for each constraint type. The objective is to maximize the total expected profit minus the total penalty.

1.2.3 Illustrative example

The illustrative example comprises 20 customers and eight activities that belong to six different campaigns. Figure 1.1 shows for each activity the day on which it takes place, the channel over which the customers are contacted, and the target products that are promoted. Table 1.1 additionally lists the cost per assignment for each activity. Table 1.2 specifies the expected profit and the response probability for each possible assignment. The business constraints comprise a maximum assignment constraint, which ensures that at most 11 *calls* are used to contact the customers, a budget constraint, which ensures that all assignments associated with the channels *direct mail*, *email*, and *text message* do not

Table 1.2: Expected profits and response probabilities of the illustrative example

Customer	Activity	Exp. profit	Response prob.	Customer	Activity	Exp. profit	Response prob.
1	A3	105	0.29	**11**	**A5**	**108**	**0.25**
1	**A4**	**105**	**0.29**	12	A3	85	0.17
1	A5	88	0.16	**12**	**A4**	**85**	**0.17**
2	A5	78	0.17	**12**	**A5**	**106**	**0.25**
2	**A6**	**91**	**0.23**	13	A3	88	0.18
3	A5	88	0.28	**13**	**A4**	**88**	**0.18**
3	**A6**	**75**	**0.20**	**13**	**A5**	**110**	**0.24**
4	A5	80	0.14	14	A5	65	0.17
4	**A6**	**90**	**0.26**	14	A7	75	0.23
5	A3	91	0.19	**14**	**A8**	**80**	**0.24**
5	**A4**	**91**	**0.19**	**15**	**A1**	**102**	**0.25**
5	**A5**	**107**	**0.26**	15	A2	102	0.25
6	**A1**	**75**	**0.22**	**15**	**A6**	**93**	**0.21**
6	A2	75	0.22	16	A3	112	0.30
6	**A6**	**85**	**0.15**	**16**	**A4**	**112**	**0.30**
7	A5	100	0.24	**16**	**A5**	**90**	**0.20**
7	**A7**	**85**	**0.11**	17	A5	89	0.29
7	**A8**	**71**	**0.22**	**17**	**A6**	**77**	**0.21**
8	A3	109	0.25	**18**	**A1**	**101**	**0.29**
8	**A4**	**109**	**0.25**	18	A2	101	0.29
8	**A5**	**91**	**0.18**	**18**	**A6**	**95**	**0.19**
9	A5	76	0.15	19	A5	68	0.20
9	**A6**	**89**	**0.25**	**19**	**A7**	**73**	**0.21**
10	A5	102	0.22	**19**	**A8**	**85**	**0.26**
10	**A7**	**87**	**0.12**	20	A5	66	0.19
10	**A8**	**68**	**0.23**	**20**	**A7**	**76**	**0.22**
11	A3	90	0.18	**20**	**A8**	**83**	**0.25**
11	**A4**	**90**	**0.18**				

incur costs of more than 40 Euros, and a maximum sales constraint, which ensures that the number of expected sales for the target product *mobile* does not exceed five. Moreover, minimum contact constraints are based on a contact rule that states that each customer must be assigned at least once. Maximum contact constraints are based on a contact rule that states that each customer must be assigned at most twice. Table 1.3 specifies for the business constraints and the contact rules the type, the start day, the end day, the channels, the target products, and the bound associated with the respective constraint or contact rule. The illustrative example has four conflict rules. First, each customer cannot be contacted more than once within two days. Second, each customer cannot be contacted more than once within five days via a *call*. Third, each customer cannot be contacted more than once within four days via *direct mail*. Fourth, each customer cannot be contacted more than once within four days via a *text message*. Table 1.4 indicates the combinations of channels and target products that lead to a conflict and the time

Table 1.3: Business constraints and contact rules of the illustrative example

Index	Type	Start day	End day	Channels	Target products	Bound
1	Maximum assignment	1	7	*call center*	ALL	11
2	Budget	1	7	*direct mail*, *email*, *text message*	ALL	40
3	Maximum sales	1	7	ALL	*mobile*	5
4	Minimum contact	1	7	ALL	ALL	1
5	Maximum contact	1	7	ALL	ALL	2

Table 1.4: Conflict rules of the illustrative example

Index	Channel 1	Target product 1	Channel 2	Target product 2	Lag
1	ALL	ALL	ALL	ALL	2
2	*call center*	ALL	*call center*	ALL	5
3	*direct mail*	ALL	*direct mail*	ALL	4
4	*text message*	ALL	*text message*	ALL	4

lag related to a conflict rule. In Tables 1.3 and 1.4, the entry "ALL" indicates that all channels or target products are affected by this constraint or rule.

All constants used to compute the total penalty for the soft constraints are set to 112 Euros (maximum absolute expected profit). The optimal assignment is marked in bold in Table 1.2 and produces an expected profit of 2,973 Euros. The maximum sales constraint is violated by 0.12, which leads to a penalty of $112(0.12) = 13.44$ Euros. The objective function value in the optimal solution is thus 2,959.56 Euros.

1.3 Literature

The literature review is organized as follows. In Section 1.3.1, we focus on related problems in direct marketing. In Section 1.3.2, we focus on more general combinatorial optimization problems that share specific constraints with our planning problem. Table 1.5 gives an overview of the discussed approaches and shows which problem features are exactly (✓) or partially ((✓)) covered.

1.3.1 Related problems in direct marketing

There are two large streams of literature in direct marketing. The first stream is concerned with the development of response models that predict the responses of customers to direct marketing efforts (e.g., Ma et al., 2016 and Lessmann et al., 2021). The second stream

pertains to the optimization of direct marketing operations given the output of response models. We focus on the second stream because it is more closely related to our planning problem.

Many of the planning problems considered in direct marketing involve the assignment of individual customers to product offers. The objective in these planning problems is to maximize the expected profit subject to various side constraints. Table 1.5 shows the side constraints that are covered by the different approaches. Cohen (2004) is the first to propose a binary linear program for assigning customers to direct marketing campaigns. For large-scale instances, he introduces a heuristic that is based on the idea of grouping similar customers and using a linear program to determine how many customers of a group are assigned to a campaign. Bhaskar et al. (2009) formulate a similar problem as the one of Cohen (2004) as a binary linear program and propose a heuristic that builds on the idea of grouping customers. Sundararajan et al. (2011) describe the integration of the heuristic of Bhaskar et al. (2009) in a retail bank, which led to an estimated financial benefit of $20 million. Nobibon et al. (2011) consider a planning problem in which a subset of products must be selected for a campaign and the customers must be assigned to the selected products. They provide a binary linear formulation, a branch-and-price algorithm, and eight heuristics for this planning problem. The two heuristics that are based on column generation and on tabu search tend to perform best. For the same planning problem as Nobibon et al. (2011), Oliveira et al. (2015) develop a heuristic that is based on a greedy randomized adaptive search procedure combined with a variable neighborhood search, and Cetin and Alabas-Uslu (2017) propose two heuristics, which in a first step use a rule-based procedure to select the products that will be part of a campaign, and in a second step assign the customers to the selected products. These two-step heuristics are competitive with the best heuristics of Nobibon et al. (2011). Finally, Coelho et al. (2017) develop a metaheuristic for a variant of the planning problem considered by Nobibon et al. (2011) where the objective function includes a reward-to-variability indicator, which is inspired by the Sharpe ratio. One major difference between our planning problem (cf. Section 1.2) and the ones discussed above is the existence of a temporal dimension. In our planning problem, each activity is scheduled on a specific day of the time horizon. This timing information is relevant for various business constraints, and especially for the conflict constraints. In the direct marketing literature, only few planning problems include a temporal dimension. The planning problems which involve a temporal dimension, however, focus primarily on the design of campaigns rather than the assignment of individual customers.

Nair and Tarasewhich (2003), for example, study a planning problem that consists of

Table 1.5: Approaches and problem features from related literature

		Approach		Time	Dec.	Obj.	Constraints				
	Authors	Exact	Heuristic	Temporal dim.	Customer asgmt	Expected profit	Min/Max asgmt	Budget	Min/Max sales	Min/Max contact	Conflict
Direct marketing	Cohen (2004)	✓	✓		✓	✓	(✓)/✓	✓		/(✓)	
	Bhaskar et al. (2009)	✓	✓		(✓)	✓		✓	(✓)/		
	Nobibon et al. (2011)	✓	✓		✓	✓	(✓/✓)	✓		/✓	
	Oliveira et al. (2015)		✓		✓	✓	(✓/✓)	✓		/✓	
	Cetin and Alabas-Uslu (2017)		✓		✓	✓	(✓/✓)	✓		/✓	
	Coelho et al. (2017)	✓	✓		✓	(✓)	(✓/✓)	✓		/✓	
	Nair and Tarasewhich (2003)	✓	✓	✓							(✓)
	Delanote et al. (2013)	✓		✓	(✓)	✓	/(✓)	✓	(✓)/	/(✓)	
	Ma and Fildes (2017)	✓	✓	✓		✓	(✓/✓)			/(✓)	
General	Darmann et al. (2011)	✓					(✓/✓)			(✓/✓)	(✓)
	Öncan et al. (2019)	✓					(✓/✓)			(✓/✓)	(✓)
	Elhedhli et al. (2011)	✓						(✓)		(✓/✓)	(✓)
	Sadykov and Vanderbeck (2013)	✓						(✓)		(✓/✓)	(✓)
	Bigler et al. (2019)	✓		✓	✓	✓	(✓)/✓	✓	(✓/✓)	(✓)/✓	✓
	This paper	✓	✓	✓	✓	✓	✓/✓	✓	✓/✓	✓/✓	✓

designing a series of promotions over time. Some of the side constraints are similar to the conflict constraints from our planning problem. These similar constraints ensure that selected pairs of promotions cannot take place within a certain number of consecutive days in the time horizon. Nair and Tarasewhich (2003) formulate a non-linear integer program and develop a genetic algorithm for this planning problem. Delanote et al. (2013) formulate an integer linear model for the planning of multi-round direct marketing campaigns, which consists of determining how many customers of each customer segment are assigned to which product and which channel in each round, in order to maximize the total expected profit. Customers who react positively in one round cannot be assigned to the same product in later rounds. The number of customers who react positively is estimated by multiplying the response probability of a segment with the respective number of assigned customers. Ma and Fildes (2017) formulate a non-linear program and develop a genetic algorithm for the planning of multi-period promotions, which consists of determining for each period which products to advertise such that the total expected profit of the campaign is maximized. Bigler et al. (2019) study a variant of the planning

problem from Section 1.2, in which all constraints are hard constraints. The authors formulate a binary linear program for this slightly different planning problem and apply it to four small- to medium-sized instances and one large-sized instance. To the best of our knowledge, no other customer assignment approach solved instances of similar size. However, this binary linear program does not scale to very large real-world instances.

As we can see from Table 1.5, none of the discussed planning problems fully cover all the features of our planning problem. For example, most planning problems do not consider conflict constraints. Conflict constraints, however, have received considerable attention as an extension of more general combinatorial optimization problems such as the assignment problem and the bin packing problem. In the next section, we review these planning problems.

1.3.2 More general combinatorial optimization problems with conflict constraints

More general combinatorial optimization problems such as the assignment problem and the bin packing problem have been extended to consider conflict constraints. In the assignment problem, equal numbers of agents (here customers) and tasks (here activities) are given, and exactly one agent must be assigned to each task such that the total cost is minimized. In the assignment problem with conflict constraints (APC), an assignment of an agent to a task may conflict with another assignment of an agent to a task. This structure of the conflict constraints is visualized in Figure 1.2. The dashed lines correspond to potential assignments of agents to tasks, the solid lines correspond to assignments of agents to tasks in a feasible solution, and the bold red line indicates a conflict. Because of the conflict, agent i_2 cannot be assigned to task j_2 when agent i_1 is assigned to task j_1, or vice versa. Darmann et al. (2011) prove that the APC is an NP-hard optimization problem. Öncan et al. (2019) propose a branch-and-bound and a Russian doll search algorithm for the APC. Figure 1.2 also illustrates the structure of the conflict constraints in this paper. The customers are eligible for some, but generally not all activities. Thus, a potential assignment in the context of our planning problem means that customer i is eligible for activity j. Other than in the APC, a customer can be assigned to multiple activities. The conflicts in this paper occur between activities and apply to all customers who are eligible for the conflicting activities. For example, a conflict exists between activities j_1 and j_2, and thus both customers i_1 and i_2 can at most be assigned to one of the two activities. In the feasible assignment shown in Figure 1.2, customer i_1 is assigned only to activity j_1 and customer i_2 is assigned only to activity j_2. An infeasible assignment

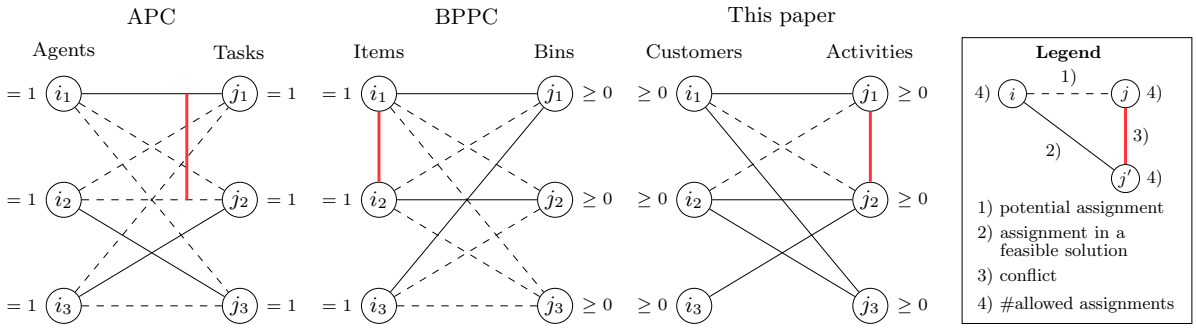

Figure 1.2: Conflict constraints in more general combinatorial optimization problems

would be, for example, if customer i_1 was assigned to both activities j_1 and j_2.

In the bin packing problem, items (here customers) of different size must be packed in a minimum number of identical bins (here activities) with limited capacity. Each item must be assigned to exactly one bin, and each bin may contain multiple items that do not exceed its bin capacity. In the bin packing problem with conflict constraints (BPPC), conflicts occur between items and apply to all bins. This structure is also illustrated in Figure 1.2, where items i_1 and i_2 are in conflict and thus cannot be assigned to the same bin. Elhedhli et al. (2011) and Sadykov and Vanderbeck (2013) develop different branch-and-price algorithms for the BPPC. Our planning problem differs from the BPPC in several ways. First, each item in the BPPC can be assigned to each bin if the capacity of the bin allows it. Second, while in our planning problem the conflicts occur between activities and apply to all customers who are eligible for these conflicting activities, the conflicts in the BPPC occur between items (here customers) and apply to all bins (here activities). Even if we consider items to be bins and bins to be items in the BPPC, there is no direct analogy to our planning problem because items must be assigned exactly once in the BPPC, while customers can be assigned multiple times and activities can have multiple assigned customers in our planning problem. Thus, the conflict constraints have the same structure only if we adjust our planning problem such that all customers are eligible for all activities and each activity must have exactly one assigned customer.

Also other planning problems such as the knapsack problem with conflict constraints (cf. Bettinelli et al., 2017 and Coniglio et al., 2021), the maximum flow problem with conflict constraints (cf. Şuvak et al., 2020), and the transportation problem with conflict constraints (cf. Sun, 2002) have attracted considerable attention in the literature. However, these planning problems differ considerably from our planning problem. In the knapsack problem, a conflict constraint ensures that only one of two items is included in the knapsack while in our planning problem a conflict constraint ensures that a customer

is assigned to at most one of two conflicting activities. In the maximum flow problem, a conflict can involve any two edges of a graph while in our planning problem, a conflict always involves the assignments of one customer to two conflicting activities. In the transportation problem, a conflict constraint ensures that two conflicting goods cannot be shipped to the same warehouse. These conflict constraints are structurally similar to our conflict constraints if we consider warehouses to be customers and goods to be activities. However, in the transportation problem with conflict constraints, the supply of goods is given while in our planning problem the number of assignments to the activities is to be determined. Also, the warehouses in the transportation problem do not need to have a minimum number of assigned goods while the customers in our planning problem can be affected by minimum contact constraints. Finally, the assignment decisions in the transportation problem are integer (quantities of goods) while in our planning problem the assignment decisions are binary.

Table 1.5 also shows some side constraints besides the conflict constraints that the approaches for the APC and the BPPC cover. However, we can see that none of the existing approaches cover all problem features of our planning problem.

1.4 Mixed-binary linear programming formulation

In this section, we formulate the planning problem as a mixed-binary linear programming formulation (MBLP). Table 1.6 summarizes the sets, parameters and decision variables of the MBLP. The superscripts (i.e., $\underline{a}$, $\overline{a}$, $\overline{b}$, $\underline{m}$, $\overline{m}$, $\underline{s}$, $\overline{s}$) indicate for which type of constraint a set, parameter, or decision variable has been introduced. The information required to generate these sets and parameters for a specific instance can be derived from four tables. These four tables are exemplarily provided for the illustrative example in Section 1.2.3 (cf. Tables 1.1, 1.2, 1.3, and 1.4). From the equivalent of Table 1.1, we can derive information about all activities. From the equivalent of Table 1.2, we can derive information on all eligible customers for each activity. From the equivalent of Table 1.3, we can identify the activities that are associated with the business constraints and the contact rules. Based on the equivalents of Tables 1.1 and 1.4, we can identify all pairwise conflicts between activities. The MBLP uses binary decision variables x_{ij}, which take the value of one if customer i is assigned to activity j, and zero otherwise. Furthermore, continuous slack variables are introduced for all soft constraints. The expected profit e_{ij} is computed by multiplying the change in revenue that results if customer i accepts the offer times the corresponding response probability q_{ij} minus the cost per assignment c_j. The MBLP reads as follows:

Table 1.6: Notation of MBLP

Sets	
I	Customers
J	Activities
T	Pairs of conflicting activities
I_j	Eligible customers of activity j
J_i	Activities for which customer i is eligible
$J_l^{\underline{a}}$	Activities associated with minimum assignment constraint $l = 1, \ldots, n^{\underline{a}}$
$J_l^{\overline{a}}$	Activities associated with maximum assignment constraint $l = 1, \ldots, n^{\overline{a}}$
$J_l^{\overline{b}}$	Activities associated with budget constraint $l = 1, \ldots, n^{\overline{b}}$
$J_l^{\underline{m}}$	Activities associated with minimum contact rule $l = 1, \ldots, n^{\underline{m}}$
$J_l^{\overline{m}}$	Activities associated with maximum contact rule $l = 1, \ldots, n^{\overline{m}}$
$J_l^{\underline{s}}$	Activities associated with minimum sales constraint $l = 1, \ldots, n^{\underline{s}}$
$J_l^{\overline{s}}$	Activities associated with maximum sales constraint $l = 1, \ldots, n^{\overline{s}}$
Parameters	
$b_l^{\underline{a}}$	Lower bound of minimum assignment constraint $l = 1, \ldots, n^{\underline{a}}$
$b_l^{\overline{a}}$	Upper bound of maximum assignment constraint $l = 1, \ldots, n^{\overline{a}}$
$b_l^{\overline{b}}$	Upper bound of budget constraint $l = 1, \ldots, n^{\overline{b}}$
$b_l^{\underline{m}}$	Lower bound of minimum contact rule $l = 1, \ldots, n^{\underline{m}}$
$b_l^{\overline{m}}$	Upper bound of maximum contact rule $l = 1, \ldots, n^{\overline{m}}$
$b_l^{\underline{s}}$	Lower bound of minimum sales constraint $l = 1, \ldots, n^{\underline{s}}$
$b_l^{\overline{s}}$	Upper bound of maximum sales constraint $l = 1, \ldots, n^{\overline{s}}$
c_j	Cost per assignment to activity j
e_{ij}	Expected profit of customer i when assigned to activity j
$n^{\underline{a}}$	Number of minimum assignment constraints
$n^{\overline{a}}$	Number of maximum assignment constraints
$n^{\overline{b}}$	Number of budget constraints
$n^{\underline{m}}$	Number of minimum contact rules
$n^{\overline{m}}$	Number of maximum contact rules
$n^{\underline{s}}$	Number of minimum sales constraints
$n^{\overline{s}}$	Number of maximum sales constraints
q_{ij}	Response probability of customer i when assigned to activity j
α	Constant to penalize the extent to which bound in a minimum assignment constraint is violated
β	Constant to penalize the extent to which bound in a minimum sales constraint is violated
γ	Constant to penalize the extent to which bound in a maximum sales constraint is violated
δ	Constant to penalize the extent to which bound in a constraint resulting from a minimum contact rule is violated
Decision variables	
x_{ij}	$= 1$, if customer i is assigned to activity j; $= 0$, otherwise
$z_l^{\underline{a}} \in [0, b_l^{\underline{a}}]$,	Slack variable of minimum assignment constraint $l = 1, \ldots, n^{\underline{a}}$
$z_l^{\underline{s}} \in [0, b_l^{\underline{s}}]$,	Slack variable of minimum sales constraint $l = 1, \ldots, n^{\underline{s}}$
$z_l^{\overline{s}} \in [0, q]$,	Slack variable of maximum sales constraint $l = 1, \ldots, n^{\overline{s}}$ with $q = \sum_{j \in J} \sum_{i \in I_j} q_{ij}$
$z_{il}^{\underline{m}} \in [0, b_l^{\underline{m}}]$,	Slack variable of minimum contact rule $l = 1, \ldots, n^{\underline{m}}$ for customer $i \in I$

$$
\text{(MBLP)} \left\{
\begin{array}{lll}
\text{Max.} & \displaystyle\sum_{j\in J}\sum_{i\in I_j} e_{ij}x_{ij} - (\alpha \sum_{l=1}^{n^{\underline{a}}} z_l^{\underline{a}} + \beta \sum_{l=1}^{n^{\underline{s}}} z_l^{\underline{s}} + \gamma \sum_{l=1}^{n^{\overline{s}}} z_l^{\overline{s}} + \delta \sum_{i\in I}\sum_{l=1}^{n^{\underline{m}}} z_{il}^{\underline{m}}) & (1)\\
\text{s.t.} & \displaystyle\sum_{j\in J_l^{\underline{a}}}\sum_{i\in I_j} x_{ij} + z_l^{\underline{a}} \geq b_l^{\underline{a}} \quad (l = 1,\ldots,n^{\underline{a}}) & (2)\\
& \displaystyle\sum_{j\in J_l^{\overline{a}}}\sum_{i\in I_j} x_{ij} \leq b_l^{\overline{a}} \quad (l = 1,\ldots,n^{\overline{a}}) & (3)\\
& \displaystyle\sum_{j\in J_l^{\overline{b}}}\sum_{i\in I_j} c_j x_{ij} \leq b_l^{\overline{b}} \quad (l = 1,\ldots,n^{\overline{b}}) & (4)\\
& \displaystyle\sum_{j\in J_l^{\underline{s}}}\sum_{i\in I_j} q_{ij}x_{ij} + z_l^{\underline{s}} \geq b_l^{\underline{s}} \quad (l = 1,\ldots,n^{\underline{s}}) & (5)\\
& \displaystyle\sum_{j\in J_l^{\overline{s}}}\sum_{i\in I_j} q_{ij}x_{ij} - z_l^{\overline{s}} \leq b_l^{\overline{s}} \quad (l = 1,\ldots,n^{\overline{s}}) & (6)\\
& \displaystyle\sum_{j\in J_l^{\underline{m}}\cap J_i} x_{ij} + z_{il}^{\underline{m}} \geq b_l^{\underline{m}} \quad (i\in I;\ l = 1,\ldots,n^{\underline{m}}) & (7)\\
& \displaystyle\sum_{j\in J_l^{\overline{m}}\cap J_i} x_{ij} \leq b_l^{\overline{m}} \quad (i\in I;\ l = 1,\ldots,n^{\overline{m}}:\ |J_l^{\overline{m}}\cap J_i| > b_l^{\overline{m}}) & (8)\\
& x_{ij_1} + x_{ij_2} \leq 1 \quad ((j_1,j_2)\in T;\ i\in I_{j_1}\cap I_{j_2}) & (9)\\
& x_{ij} \in \{0,1\} \quad (j\in J;\ i\in I_j) & (10)\\
& z_l^{\underline{a}} \in [0, b_l^{\underline{a}}] \quad (l = 1,\ldots,n^{\underline{a}}) & (11)\\
& z_l^{\underline{s}} \in [0, b_l^{\underline{s}}] \quad (l = 1,\ldots,n^{\underline{s}}) & (12)\\
& z_l^{\overline{s}} \in [0, q] \quad (l = 1,\ldots,n^{\overline{s}}) & (13)\\
& z_{il}^{\underline{m}} \in [0, b_l^{\underline{m}}] \quad (i\in I;\ l = 1,\ldots,n^{\underline{m}}) & (14)
\end{array}
\right.
$$

The objective function (1) is a linear combination of the total expected profit and the total penalty. The total expected profit corresponds to the sum over all expected profits e_{ij} that result from assigning a customer i to an activity j. The total penalty corresponds to the sum of the products of the slack variables and the corresponding penalty constants α, β, γ, or δ. Constraints (2) represent the minimum assignment constraints. For each minimum assignment constraint l, the number of customers assigned to the relevant activities $J_l^{\underline{a}}$ plus the corresponding slack variable $z_l^{\underline{a}}$ must satisfy the lower bound $b_l^{\underline{a}}$. Constraints (3) correspond to the maximum assignment constraints. For each maximum assignment constraint l, the number of customers assigned to the relevant activities $J_l^{\overline{a}}$ must not exceed the upper bound $b_l^{\overline{a}}$. Constraints (4) represent the budget constraints. Each assignment of a customer i to an activity j generates a cost c_j. For each budget constraint l, a budget $b_l^{\overline{b}}$ is imposed on the total cost that results from assigning customers to the relevant activities $J_l^{\overline{b}}$. Constraints (5) represent the minimum sales constraints. Each assignment of a customer i to an activity j leads to a positive customer response with a

probability q_{ij}. For each minimum sales constraint l, the number of expected sales that results from assigning customers to the relevant activities $J_l^{\underline{s}}$, i.e., the sum of the corresponding response probabilities of the assigned customers, plus the corresponding slack variable $z_l^{\underline{s}}$ must satisfy the lower bound $b_l^{\underline{s}}$. Constraints (6) represent the maximum sales constraints. For each maximum sales constraint l, the number of expected sales that results from assigning customers to the relevant activities $J_l^{\overline{s}}$ minus the corresponding slack variable $z_l^{\overline{s}}$ must not exceed the upper bound $b_l^{\overline{s}}$. Constraints (7) represent the minimum contact constraints. For each contact rule $l = 1, \ldots, n^{\underline{m}}$, a separate constraint is imposed for each customer. This constraint ensures that the number of times customer i is assigned to the relevant activities $J_l^{\underline{m}} \cap J_i$ plus the corresponding slack variable $z_{il}^{\underline{m}}$ satisfies the lower bound $b_l^{\underline{m}}$. Constraints (8) correspond to the maximum contact constraints. For each contact rule $l = 1, \ldots, n^{\overline{m}}$, a separate constraint is imposed for each customer if this customer is eligible for more than $b_l^{\overline{m}}$ relevant activities $J_l^{\overline{m}} \cap J_i$. This constraint ensures that the number of times customer i is assigned to the relevant activities $J_l^{\overline{m}} \cap J_i$ does not exceed the upper bound $b_l^{\overline{m}}$. Constraints (9) represent the conflict constraints. The conflict constraints guarantee for each pair of conflicting activities (j_1, j_2) in set T and for each customer i who is eligible for both activities j_1 and j_2 that the customer can only be assigned to one of the two conflicting activities. Note that this formulation is very similar to the formulation of Bigler et al. (2019) that was developed for a slightly different planning problem. If one wants to consider all constraints as hard constraints, the upper bounds on the slack variables can be set to zero.

1.5 Matheuristic

The main idea of the matheuristic is to solve a mathematical model for groups of customers rather than individual customers. The key feature is that we incorporate customer-specific constraints in the group-level model, which allows transforming the solution from the group-level model into a customer-level solution with almost no loss in solution quality. Moreover, the matheuristic is designed in such a way that the user can control the trade-off between solution quality and running time with a single parameter. The matheuristic consists of four steps. Figure 1.3 provides an overview of these four steps. In Sections 1.5.1–1.5.4, we explain the four steps in detail. In Section 1.5.5, we apply the proposed matheuristic to the illustrative example from Section 1.2.3.

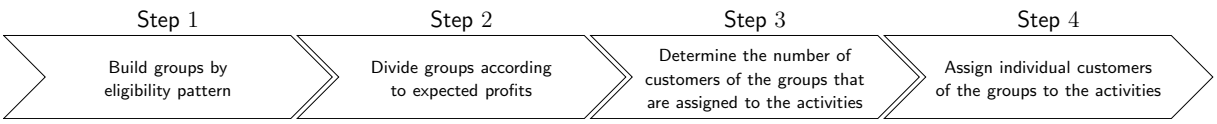

Figure 1.3: Steps of the matheuristic

1.5.1 Build groups by eligibility pattern (Step 1)

In the first step of the matheuristic, the customers are grouped according to their eligibility patterns. An eligibility pattern p indicates the activities for which a customer is eligible. Two customers share the same eligibility pattern if they are eligible for the same activities. For example, for an instance with three activities and a customer who is eligible for activities 1 and 3, but not for activity 2, the customer's eligibility pattern corresponds to $[1, 0, 1]$. We generate an eligibility matrix with $|I|$ rows and $|J|$ columns. This matrix contains values of only zero and one, where a value of one indicates that a customer $i \in I$ is eligible for an activity $j \in J$. All customers with the same eligibility pattern are grouped together. This grouping can be efficiently produced by sorting the rows of the eligibility matrix. The grouping by eligibility patterns is essential for the decomposition strategy of the matheuristic because it ensures that the customers in the same group are affected by the same conflicts between activities. Only due to this feature of the customer groups we are able to enforce customer-specific constraints in the third step of the matheuristic. Note that if the number of eligibility patterns is not much smaller than the number of customers, one can reduce the number of different eligibility patterns by grouping similar but not identical patterns and replacing all patterns within each group with the intersection of the different patterns in the group. However, according to the telecommunications company, the number of different patterns is always substantially lower than the number of customers in real-world instances because the customers with the same subscriptions (or products) are mostly eligible for the same activities.

1.5.2 Divide groups according to expected profits (Step 2)

In the second step of the matheuristic, each group is further divided into up to k subgroups. We determine the subgroups by considering a clustering problem for each group. The goal is to partition the customers of the group into k clusters (subgroups) such that the customers in the same cluster have similar expected profits for the activities for which they are eligible. Note that all customers of the same group are eligible for the same activities. Hence, the input to each clustering problem is a matrix which has one row for each customer and one column for each activity for which the customers are eligible. The

Table 1.7: Additional notation of linear model

Sets	
G	Groups determined in step 2 of the matheuristic
P	Eligibility patterns
G_p	Groups with eligibility pattern p
J_g	Activities for which customers in group g are eligible
$J_{lp}^{\overline{c}}$	Activities associated with constraint $l = 1, \ldots, n_p^{\overline{c}}$, which ensures conflict rules for groups with eligibility pattern p
Parameters	
$\overline{e}_{gj}$	Average expected profit of customers of group g when assigned to activity j
$n_p^{\overline{c}}$	Number of constraints that are set up for each group g with eligibility pattern p to ensure conflict rules
o_g	Number of customers in group g
$\overline{q}_{gj}$	Average response probability of customers of group g when assigned to activity j
Decision variables	
$x_{gj} \in [0, o_g]$,	Number of customers of group g that are assigned to activity j
$z_{gl}^{\underline{m}} \in [0, b_l^{\underline{m}} o_g]$,	Slack variable of group g for minimum contact rule $l = 1, \ldots, n^{\underline{m}}$

values in the matrix correspond to the expected profits of the customers for the respective activities. To determine the partition, we apply the mini batch k-means algorithm of Sculley (2010). The mini batch k-means algorithm is particularly suitable for large-scale applications due to its speed and memory efficiency. If there are fewer than k customers in a group, all customers are placed in separate subgroups. Increasing the parameter k leads to smaller but more homogeneous subgroups. For the sake of simplicity, the subgroups that result from this grouping step are subsequently referred to as groups of customers.

1.5.3 Determine the number of customers of the groups that are assigned to the activities (Step 3)

In the third step of the matheuristic, a linear model (LP) determines how many customers of each group are assigned to the activities. We introduce continuous decision variables x_{gj} that indicate how many customers of group g are assigned to activity j. For each decision variable, we compute the corresponding average expected profit $\overline{e}_{gj}$ and the average response probability $\overline{q}_{gj}$ based on the respective expected profits e_{ij} and response probabilities q_{ij} of the customers of group g. Table 1.7 shows the notation that is used, in addition to the notation already introduced in Table 1.6. The LP reads as follows:

$$
\text{(LP)} \left\{
\begin{array}{lll}
\text{Max.} & \displaystyle\sum_{g\in G}\sum_{j\in J_g} \overline{e}_{gj}x_{gj} - (\alpha \sum_{l=1}^{n^{\underline{a}}} z_l^{\underline{a}} + \beta \sum_{l=1}^{n^{\underline{s}}} z_l^{\underline{s}} + \gamma \sum_{l=1}^{n^{\overline{s}}} z_l^{\overline{s}} + \delta \sum_{g\in G}\sum_{l=1}^{n^{\underline{m}}} z_{gl}^{\underline{m}}) & (15)\\
\text{s.t.} & \displaystyle\sum_{g\in G}\sum_{j\in J_g\cap J_l^{\underline{a}}} x_{gj} + z_l^{\underline{a}} \geq b_l^{\underline{a}} \qquad (l=1,\ldots,n^{\underline{a}}) & (16)\\
& \displaystyle\sum_{g\in G}\sum_{j\in J_g\cap J_l^{\overline{a}}} x_{gj} \leq b_l^{\overline{a}} \qquad (l=1,\ldots,n^{\overline{a}}) & (17)\\
& \displaystyle\sum_{g\in G}\sum_{j\in J_g\cap J_l^{\overline{b}}} c_j x_{gj} \leq b_l^{\overline{b}} \qquad (l=1,\ldots,n^{\overline{b}}) & (18)\\
& \displaystyle\sum_{g\in G}\sum_{j\in J_g\cap J_l^{\underline{s}}} \overline{q}_{gj}x_{gj} + z_l^{\underline{s}} \geq b_l^{\underline{s}} \quad (l=1,\ldots,n^{\underline{s}}) & (19)\\
& \displaystyle\sum_{g\in G}\sum_{j\in J_g\cap J_l^{\overline{s}}} \overline{q}_{gj}x_{gj} - z_l^{\overline{s}} \leq b_l^{\overline{s}} \quad (l=1,\ldots,n^{\overline{s}}) & (20)\\
& \displaystyle\sum_{j\in J_g\cap J_l^{\underline{m}}} x_{gj} + z_{gl}^{\underline{m}} \geq o_g b_l^{\underline{m}} \qquad (g\in G;\ l=1,\ldots,n^{\underline{m}}) & (21)\\
& \displaystyle\sum_{j\in J_g\cap J_l^{\overline{m}}} x_{gj} \leq o_g b_l^{\overline{m}} \qquad (g\in G;\ l=1,\ldots,n^{\overline{m}}) & (22)\\
& \displaystyle\sum_{j\in J_{lp}^{\overline{c}}} x_{gj} \leq o_g \qquad (p\in P;\ g\in G_p;\ l=1,\ldots,n_p^{\overline{c}}) & (23)\\
& x_{gj} \in [0, o_g] \qquad (g\in G;\ j\in J_g) & (24)\\
& z_{gl}^{\underline{m}} \in [0, b_l^{\underline{m}} o_g] \qquad (g\in G;\ l=1,\ldots,n^{\underline{m}}) & (25)\\
& (11)-(13) &
\end{array}
\right.
$$

In the objective function (15), the total average expected profit minus the total penalty associated with the soft constraints is maximized. Constraints (16)–(22) are formulated analogously to the constraints from Section 1.4 for groups of customers instead of individual customers. In the group-level model, it is not sufficient to simply consider pairs of conflicting activities for incorporating the conflict rules. Considering only pairs of activities, as in constraints (9), will result in excessive assignments on the group level, as shown in the following example. Consider two customers i_1 and i_2 who both belong to group g_1. Assume that these customers are eligible for three activities j_1, j_2, and j_3, which all have conflicts among each other. Therefore, each customer can only be assigned to one of the three activities. The analogous pairwise constraints to the constraints (9) for this example would be formulated as follows: $x_{g_1,j_1} + x_{g_1,j_2} \leq o_{g_1}$, $x_{g_1,j_1} + x_{g_1,j_3} \leq o_{g_1}$, $x_{g_1,j_2} + x_{g_1,j_3} \leq o_{g_1}$ with $o_{g_1} = 2$. The solution $x_{g_1,j_1} = x_{g_1,j_2} = x_{g_1,j_3} = 1$ satisfies these constraints. However, because there are only two customers in group g_1 and each customer can only be assigned to one of the three activities j_1, j_2, and j_3, this solution cannot be translated into

a customer-level solution without losing one assignment. To better represent the conflict rules already in the group-level model, we introduce an alternative modeling technique. For each eligibility pattern p, we generate one or multiple sets $J_{lp}^{\bar{c}}$ of conflicting activities of maximal size. Constraints (23) ensure that a maximum of o_g customers can be assigned to two or more conflicting activities $J_{lp}^{\bar{c}}$ for each eligibility pattern p and each group g with eligibility pattern p. In Section 1.6, we will explain in detail how these sets of conflicting activities $J_{lp}^{\bar{c}}$ are efficiently generated. Note that there are still special cases in which the LP might assign too many customers on the group level. Consider five activities j_1–j_5 with $T = \{(j_1, j_2), (j_2, j_3), (j_3, j_4), (j_4, j_5), (j_5, j_1)\}$. Assume that group g_1 has two customers i_1 and i_2 who are eligible for all five activities. Then, the solution $x_{g_1,j_1} = x_{g_1,j_2} = x_{g_1,j_3} = x_{g_1,j_4} = x_{g_1,j_5} = 1$ is feasible for constraints (23). However, one of these five assignments will be lost in the customer-level assignment.

1.5.4 Assign individual customers of the groups to the activities (Step 4)

In this step, we apply an iterative algorithm to assign individual customers to the activities based on the group-level assignment from the previous step. Figure 1.4 provides a flowchart of the algorithm. The basic idea is to assign the most profitable customers of group g to activity j for each variable x_{gj} of the LP with a non-zero value. The iterative algorithm assigns the customers to the activities without violating hard constraints. Next, we explain the iterative algorithm step-by-step.

First, the algorithm determines a specific sequence for the group-activity pairs of the decision variables x_{gj} with a value greater than or equal to one. This sequence determines in which order the customers are assigned to the activities. A random sequence is likely to lead to a loss of group-level assignments, as illustrated by the following example. Consider a group g_1 that contains two customers i_1 and i_2. Assume that the customers of group g_1 are eligible for the three activities j_1, j_2, and j_3 and that activity j_1 conflicts with both activities j_2 and j_3. Further assume that the solution of the LP is $x_{g_1,j_1} = x_{g_1,j_2} = x_{g_1,j_3} = 1$ and that the following random sequence $[(g_1, j_2), (g_1, j_3), (g_1, j_1)]$ of the group-activity pairs is given. If customer i_1 is assigned to activity j_2 in the first iteration, and customer i_2 is assigned to activity j_3 in the second iteration, then no customer can be assigned to activity j_1 in the third iteration because both customers are already assigned to activities that conflict with activity j_1. Instead of using a random sequence, we prepare a sequence based on a conflict graph $\mathcal{G} = (V, E)$ which can be constructed from the conflict rules. The nodes V of the conflict graph $\mathcal{G}$ correspond to the activities of an instance (i.e.,

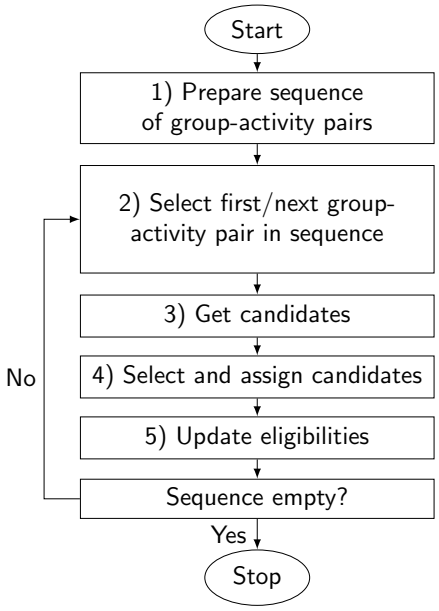

Figure 1.4: Flowchart of the iterative algorithm

$V = J$), and the edges E between nodes represent conflicts among activities. We first sort all group-activity pairs according to the group index such that all activities of the same group are processed sequentially. To determine the sequence of activities for each group g, we construct a subgraph of the conflict graph $\mathcal{G}$ that only contains the activities as nodes to which customers from group g are assigned. The activity that corresponds to the node with the highest degree in this subgraph is selected first. Then, activities are added iteratively to the sequence in decreasing order of the number of edges that connect the respective nodes in the subgraph to nodes that represent already added activities. Note that this number of edges is recomputed every time an activity is added to the sequence. In case of ties, the activity with the lower index is first. For the example from above, this sorting mechanism results in the sequence $[(g_1, j_1), (g_1, j_2), (g_1, j_3)]$, based on which all three assignments can be conducted on the customer level.

Second, the iterative algorithm selects the first/next group-activity pair (g, j) in the sequence.

Third, the iterative algorithm identifies the customers of group g that can be assigned to activity j without violating any hard customer-specific constraints. Even though all customers of group g are initially eligible for activity j, it is possible that some customers of group g cannot be assigned to activity j because such an assignment would violate a conflict rule or a maximum contact rule due to assignments in earlier iterations. The customers of group g that can be assigned are referred to as candidates.

Fourth, the iterative algorithm selects the $\lfloor x_{gj} \rfloor$ candidates with the highest expected profits for activity j and assigns them. If the number of candidates is lower than $\lfloor x_{gj} \rfloor$, then all candidates are assigned. The number of candidates can be lower than $\lfloor x_{gj} \rfloor$ in special cases. Such a special case is the example from Section 1.5.3 in which the conflict graph $\mathcal{G}$ represents a circle of five activities j_1, j_2, j_3, j_4, and j_5, i.e., set $T = \{(j_1, j_2), (j_2, j_3), (j_3, j_4), (j_4, j_5), (j_5, j_1)\}$. The last group-activity pair in this example may have no candidate because all customers of the group have been assigned to conflicting activities in previous iterations, even though the solution of the LP intended to assign one or more customers.

Fifth, the iterative algorithm updates the eligibilities of the assigned candidates for other activities. All assigned candidates are no longer eligible for activities that are in conflict with activity j. Moreover, it is possible that with the assignment to activity j, some of the candidates will reach their maximum number of assignments for one or multiple maximum contact rules. These customers are no longer eligible for other activities that are affected by these maximum contact rules. Finally, the iterative algorithm determines whether the sequence of group-activity pairs is empty. If that is the case, the iterative algorithm stops and returns the customer-level assignment x_{ij}; otherwise, the next group-activity pair is selected from the sequence.

1.5.5 Illustrative example

We apply the matheuristic to the illustrative example from Section 1.2.3 step-by-step. In the first step, the customers who are eligible for the same activities are grouped together. This leads to the following four groups: $\{7, 10, 14, 19, 20\}$, $\{2, 3, 4, 9, 17\}$, $\{1, 5, 8, 11, 12, 13, 16\}$, and $\{6, 15, 18\}$. In the second step, these groups are further divided using the mini batch k-means algorithm with a value of $k = 2$. The first two columns of Table 1.8 show the resulting eight groups and the customers I_g who belong to these groups. In the third step, the LP is set up and solved for the eight groups and eight activities. Table 1.8 shows the resulting values of the decision variables x_{gj}. A dash (-) indicates that the customers of this group are not eligible for this activity, and thus no assignment can be conducted. In the fourth step, the customers are iteratively assigned to the activities. Figure 1.5 shows the conflict graph $\mathcal{G}$ based on which the sequence of the group-activity pairs is determined. The complete sequence of the group-activity pairs is: [(1, A7), (1, A8), (2, A7), (2, A8), (3, A6), (4, A6), (5, A4), (5, A5), (6, A3), (6, A4), (6, A5), (7, A1), (7, A6), (8, A1), (8, A6)]. The algorithm iterates over all group-activity pairs in the derived order. In most of the iterations, all customers of the respective groups are assigned to the activities (as intended by the solution of the LP). Next, we will describe the iterations in which not all customers

Table 1.8: Groups in the illustrative example

		x_{gj}							
Group g	Customers I_g	$j = $ A1	$j = $ A2	$j = $ A3	$j = $ A4	$j = $ A5	$j = $ A6	$j = $ A7	$j = $ A8
1	$\{7, 10\}$	-	-	-	-	0	-	2	2
2	$\{14, 19, 20\}$	-	-	-	-	0	-	2	3
3	$\{3, 17\}$	-	-	-	-	0	2	-	-
4	$\{2, 4, 9\}$	-	-	-	-	0	3	-	-
5	$\{5, 11, 12, 13\}$	-	-	0	4	4	-	-	-
6	$\{1, 8, 16\}$	-	-	1	2	2	-	-	-
7	$\{15, 18\}$	2	0	-	-	-	2	-	-
8	$\{6\}$	1	0	-	-	-	1	-	-

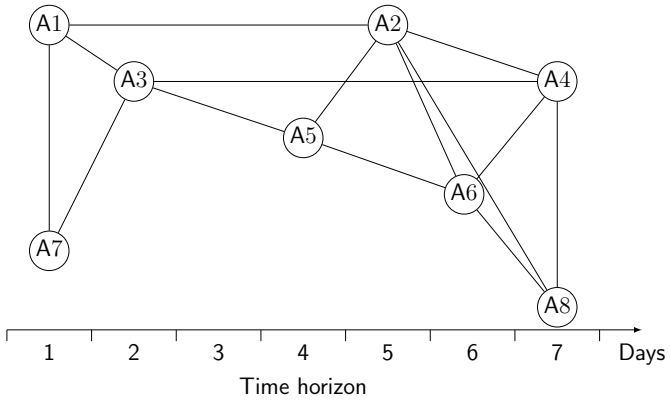

Figure 1.5: Conflict graph $\mathcal{G}$ for the illustrative example

of a group are assigned to the considered activity. In the third iteration, customers of group 2 are assigned to activity A7. All three customers of group 2 are candidates, but $\lfloor x_{2,\text{A7}} \rfloor = 2$. Thus, only the two customers with the highest expected profits for activity A7 are assigned (here, customers 20 and 14). Also, in iteration 9, only one customer of group 6 must be assigned to activity A3. Thus, customer 16 is assigned. In iterations 10 and 11, only customers 8 and 1 of group 6 are candidates because customer 16 has been assigned to a conflicting activity in a previous iteration. Thus, these two customers are assigned to both activities A4 and A5. The total expected profit of the solution obtained by the matheuristic is 2,973 Euros (the same as in the optimal solution). The maximum sales constraint is violated by 0.14 (+0.02 as compared to the optimal solution), which leads to a penalty of $112(0.14) = 15.68$ Euros. The objective function value of the solution derived by the matheuristic is thus 2,957.32 Euros, which is 2.24 Euros below the objective function value of the optimal solution.

Table 1.9: Matrices used in the preprocessing technique

Matrix	Initial dimensions	Domain	Description
A	$(c \times \lvert J \rvert)$	Binary	Contains the c maximal cliques of conflict graph $\mathcal{G}$ in rows
B	$(c' \times \lvert J \rvert)$	Binary	Contains the c' rows with two or more non-zero entries that result from an element-wise multiplication of each row of matrix **A** with eligibility pattern p
C	$(c' \times c')$	Integer	Obtained by multiplying the matrices **B** and $(\mathbf{B})^T$
D	$(c' \times c')$	Integer	Contains diagonal elements of matrix **C** in each row
E	$(c' \times c')$	Integer	Obtained by subtracting matrix **D** from matrix **C**

1.6 Preprocessing technique

In this section, we present the preprocessing technique that is used in the third step of the matheuristic in more detail. To implement constraints (23), we need to determine for each eligibility pattern $p \in P$ one or several sets of conflicting activities $J_{lp}^{\bar{c}}$. Determining these sets without introducing redundancies is non-trivial and may become a computational bottleneck for large-scale instances if not implemented in an efficient manner. We propose a preprocessing technique that combines concepts from graph theory with array-computing to efficiently generate these sets. By using arrays (matrices) as the fundamental data structure, we can benefit from highly optimized array-computing libraries. The preprocessing technique consists of six steps. In Sections 1.6.1–1.6.6, we explain each step by means of an example that involves five activities j_1, j_2, j_3, j_4, and j_5. Figure 1.6 shows an overview of the different steps of the preprocessing technique using this example. Table 1.9 provides the notation of the matrices used for the preprocessing technique. In Section 1.6.7, we introduce an alternative mixed-binary linear programming formulation that uses the sets generated by the preprocessing technique.

1.6.1 Step one of the preprocessing technique

In step 1), we generate the conflict graph $\mathcal{G}$ from the predefined conflict rules. The nodes represent the activities and the edges represent the conflicts between activities. The conflict graph of the example used in this section is shown at the top of Figure 1.6.

1.6.2 Step two of the preprocessing technique

In step 2), we identify all maximal cliques in the conflict graph $\mathcal{G}$ using the NetworkX implementation (cf. Hagberg et al., 2008) of the algorithm of Bron and Kerbosch (1973). Even though the problem of finding all maximal cliques in a graph is NP-hard, real-world

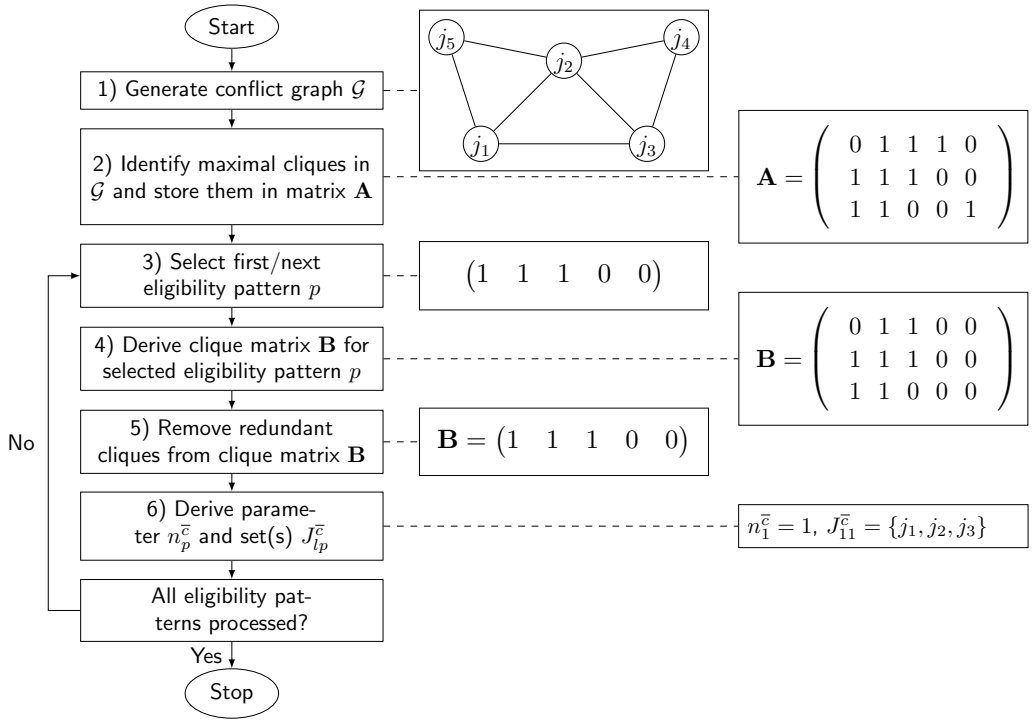

Figure 1.6: Flowchart of the preprocessing technique illustrated with an example

graphs often exhibit properties that enable solving clique problems in a few seconds, even when the graph has millions of nodes. Walteros and Buchanan (2020) found that the maximum clique problem is easy to solve on graphs with a small clique-core gap which they define to be the difference between the graph's clique number and its degeneracy-based upper bound on the clique number. Like many other real-world graphs, also the conflict graphs considered in this paper have a clique-core gap of zero. We store these maximal cliques in a binary matrix $\mathbf{A}$. Each row of matrix $\mathbf{A}$ corresponds to a maximal clique, and each column corresponds to an activity. A value of one indicates that the activity in the corresponding column is part of the maximal clique in the corresponding row. The conflict graph $\mathcal{G}$ of the example has three maximal cliques, and thus matrix $\mathbf{A}$ consists of $c = 3$ rows and $|J| = 5$ columns (cf. Figure 1.6).

1.6.3 Step three of the preprocessing technique

In step 3), we select the first/next eligibility pattern p from set P. For illustrative purposes, assume that we select an eligibility pattern $[1, 1, 1, 0, 0]$ with index $p = 1$. All customers with this eligibility pattern are eligible for the three activities j_1, j_2, and j_3 but not for the activities j_4 and j_5.

1.6.4 Step four of the preprocessing technique

In step 4), we derive the clique matrix $\mathbf{B}$ for the selected eligibility pattern p. The clique matrix $\mathbf{B}$ is computed by an element-wise multiplication of each row of matrix $\mathbf{A}$ with the eligibility pattern p. Only rows that contain two or more non-zero entries are relevant for setting up constraints to ensure conflict rules, because a conflict must always occur between at least two activities. Thus, we remove all rows that contain less than two non-zero entries. In the example, the resulting matrix $\mathbf{B}$ has $c' = 3$ rows (cf. Figure 1.6).

1.6.5 Step five of the preprocessing technique

In step 5), we remove cliques (rows) from the clique matrix $\mathbf{B}$ that are a subset of another clique (row) of matrix $\mathbf{B}$. Here, a clique is a subset of another clique if it is either identical to the other clique or if it is contained in the other clique. A clique is contained in another clique if it has only values of one in columns in which the other clique also has values of one. In the example, the first and third cliques of matrix $\mathbf{B}$ are a subset of the second clique (cf. Figure 1.6). We detect all cliques which are a subset of another clique using array-computing steps. These steps are illustrated for the matrix $\mathbf{B}$ in Figure 1.7. We start by multiplying matrix $\mathbf{B}$ with the transposed matrix $(\mathbf{B})^T$. The resulting integer matrix $\mathbf{C}$ indicates, in the diagonal, how many values of one the corresponding clique in the matrix $\mathbf{B}$ contains. The off-diagonal elements indicate, for each pair of cliques of the matrix $\mathbf{B}$, how many values of one they have in common (how many values of one occur in the same columns). For the example, matrix $\mathbf{C}$ is displayed in Figure 1.7 (cf. ①). Moreover, we generate a matrix $\mathbf{D}$ that contains the diagonal elements of matrix $\mathbf{C}$ in the rows. In the example, the first and the third rows of matrix $\mathbf{D}$ correspond to twos, and the second row of matrix $\mathbf{D}$ corresponds to threes. We then subtract matrix $\mathbf{D}$ from matrix $\mathbf{C}$ and store the values in an integer matrix $\mathbf{E}$ (cf. ② in Figure 1.7). The elements of matrix $\mathbf{E}$ show for each pair of cliques of the matrix $\mathbf{B}$ if the clique is contained in the other clique. For example, the element in the first row and second column of matrix $\mathbf{E}$ is zero, which means that the first clique of the matrix $\mathbf{B}$ is contained in the second clique of the matrix $\mathbf{B}$. Naturally, the diagonal elements of the matrix $\mathbf{E}$ are zero, because each clique of the matrix $\mathbf{B}$ is identical to itself. We then want to remove the cliques that are a subset of another clique, i.e., the rows that contain a value of zero in an off-diagonal element of the matrix $\mathbf{E}$. Here it is important to notice that if two cliques of the matrix $\mathbf{B}$ were exactly identical, then both of the corresponding rows in matrix $\mathbf{E}$ would contain a value of zero as an off-diagonal element, but we only want to remove one of these rows and keep the other one (otherwise we would miss a clique). To avoid removing too many

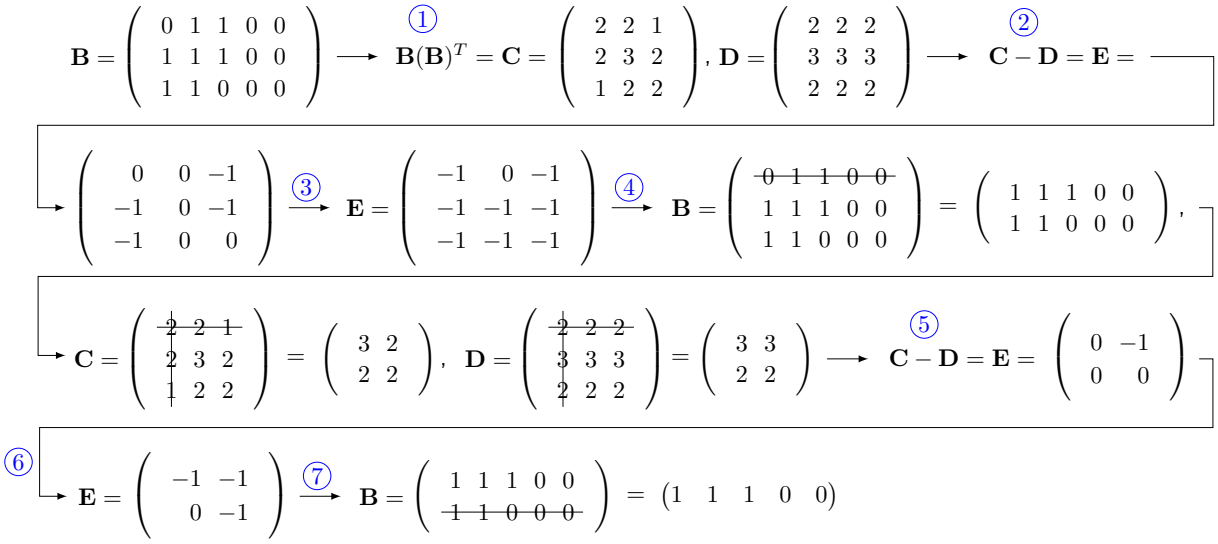

Figure 1.7: Example for removing cliques from the binary matrix **B** that are a subset of another clique

cliques, we first check whether each clique in matrix **B** is a subset of a clique *below* it. Therefore, we fill the lower half and the diagonal of matrix **E** with values of negative one (cf. ③ in Figure 1.7). Note that we could fill in any arbitrary non-zero value. Next, we check which rows in matrix **E** contain a value of zero. The rows that contain a value of zero in matrix **E** indicate that the corresponding clique in matrix **B** is a subset of another clique and can be removed. In the example, we remove the first clique from matrix **B** (cf. ④ in Figure 1.7). We also update the matrices **C** and **D** by removing the first row and first column. Next, we check whether each clique of matrix **B** is a subset of a clique *above* it. We continue with the updated integer matrix **C** and again subtract the updated matrix **D** to obtain the updated matrix **E** (cf. ⑤ in Figure 1.7). Then, we again fill the diagonal of matrix **E** with values of negative one (the upper half of matrix **E** no longer contains any zeros because these rows have already been removed) and determine whether any values of zero occur in the lower half of matrix **E** (cf. ⑥ in Figure 1.7). The rows that contain a value of zero in matrix **E** again indicate that the corresponding clique in matrix **B** is a subset of another clique and can be removed. In the example, we remove the second clique from matrix **B** (cf. ⑦ in Figure 1.7).

1.6.6 Step six of the preprocessing technique

In step 6), we derive the parameter $n_p^{\bar{c}}$ and the set(s) $J_{lp}^{\bar{c}}$ from matrix **B**. Each row of matrix **B** results in a constraint for the customers with eligibility pattern p. In the

example, matrix $\mathbf{B}$ ultimately contains one row, and thus $n_1^{\bar{c}} = 1$. Set $J_{lp}^{\bar{c}}$ corresponds to the activities associated with constraint $l = 1, \ldots, n_p^{\bar{c}}$. In the example, $J_{11}^{\bar{c}} = \{j_1, j_2, j_3\}$ because the first row of matrix $\mathbf{B}$ includes activities j_1, j_2, and j_3. Finally, we verify whether all eligibility patterns have been processed. If that is the case, we stop; otherwise, we select the next eligibility pattern p. For large-scale instances, steps 3) to 6) of Figure 1.6 can be parallelized for different eligibility patterns to further reduce running times.

As an alternative to applying our preprocessing technique, the sets $J_{lp}^{\bar{c}}$ could be determined by computing an individual conflict graph for each eligibility pattern and by deriving the sets $J_{lp}^{\bar{c}}$ from these conflict graphs directly using the algorithm of Bron and Kerbosch (1973) on each of these conflict graphs. However, constructing a separate conflict graph for each eligibility pattern and computing the maximal cliques in each of these conflict graphs is much slower than the array-computing, especially for instances comprising a large number of eligibility patterns. Here, it is important to note that our preprocessing technique is not an alternative algorithm to compute maximal cliques but rather a procedure that generates the sets of conflicting activities $J_{lp}^{\bar{c}}$ for each eligibility pattern p without introducing redundancies; i.e., cliques that are a subset of another clique for a specific eligibility pattern p are identified efficiently, which prevents introducing a large number of redundant constraints (see Tables 1.12 and 1.14).

1.6.7 An alternative mixed-binary linear programming formulation

The parameters $n_p^{\bar{c}}$ and sets $J_{lp}^{\bar{c}}$ can also be used in a mixed-binary linear programming formulation to incorporate the conflict rules. Constraints (26) ensure that each customer i with eligibility pattern p is assigned to at most one of two or more conflicting activities $J_{lp}^{\bar{c}}$. Set I_p includes all customers with the eligibility pattern p.

$$\sum_{j \in J_{lp}^{\bar{c}}} x_{ij} \leq 1 \qquad (p \in P, i \in I_p, l = 1, \ldots, n_p^{\bar{c}}) \tag{26}$$

We formulate an alternative mixed-binary linear program that uses constraints (26) instead of constraints (9) and reads as follows:

$$(\text{MBLP}') \begin{cases} \text{Max. } (1) \\ \text{s.t.} \quad (2)\text{–}(8),\ (10)\text{–}(14),\ (26) \end{cases}$$

With constraints (26), fewer constraints are required to ensure the conflict rules without loss of generality. One advantage that we noticed is that for our problem instances the

linear programming relaxation of model MBLP′ was tighter than the linear programming relaxation of model MBLP.

1.7 Results

In this section, we compare the performance of the matheuristic to the performance of the MBLP and MBLP′. In Section 1.7.1, we describe the generated and real-world instances. In Section 1.7.2, we present the experimental design. In Section 1.7.3, we compare the performance of the MBLP to the performance of the MBLP′. In Section 1.7.4, we assess the overall performance of our matheuristic and investigate the effect of two key components of the matheuristic on running time and solution quality.

1.7.1 Problem instances

Our test set comprises 13 real-world instances and 27 instances that we manually generated based on real-world data (cf. Table 1.10). First, we describe the generated instances in more detail. These instances include small (GS), medium (GM), and large (GL) instances. The small instances comprise up to 20,000 customers and 75 activities, the medium instances comprise up to 200,000 customers and 125 activities, and the large instances comprise up to 1,000,000 customers and 175 activities. We generated different instances by varying the eligibility fraction (small/large) and the number of eligibility patterns (few/many). The eligibility fraction specifies the percentage of activities a customer is eligible for on average. For the generated instances, the eligibility fraction in Table 1.10 might slightly differ from the actual eligibility fraction of the instance as a consequence of the randomized generation process. The response probabilities, the costs per assignment, and the expected profits were derived from real-world data. The costs per assignment and the expected profits were scaled with a factor to preserve confidentiality. The constraints were defined for each instance in consultation with the company. All generated instances are publicly available (cf. `https://github.com/phil85/customer-assignment-instances`). The real-world instances consist of five large instances (RL) comprising up to 1.4 million customers and 385 activities, and eight very large instances (RVL) comprising over 2 million customers and up to 295 activities. While the RL instances have small eligibility fractions but many eligibility patterns, the RVL instances have high eligibility fractions but few eligibility patterns. The real-world instances are confidential and thus not publicly available. The instances GS1', GM1', GL1', RL1', RVL1' differ from the respective instances GS1, GM1,

Table 1.10: Generated and real-world problem instances

ID	Customers	Activities	Eligibility fraction [%]	Eligibility patterns
GS1	10,000	50	small (5)	few (50)
GS1'	10,000	50	small (5)	few (50)
GS2	10,000	50	large (15)	few (50)
GS3	10,000	50	small (5)	many (100)
GS4	10,000	50	large (15)	many (100)
GS5	20,000	75	small (5)	few (50)
GS6	20,000	75	large (15)	few (50)
GS7	20,000	75	small (5)	many (100)
GS8	20,000	75	large (15)	many (100)
GM1	100,000	100	small (5)	few (300)
GM1'	100,000	100	small (5)	few (300)
GM2	100,000	100	large (15)	few (300)
GM3	100,000	100	small (5)	many (800)
GM4	100,000	100	large (15)	many (800)
GM5	200,000	125	small (5)	few (300)
GM6	200,000	125	large (15)	few (300)
GM7	200,000	125	small (5)	many (800)
GM8	200,000	125	large (15)	many (800)
GL1	500,000	150	small (5)	few (300)
GL1'	500,000	150	small (5)	few (300)
GL2	500,000	150	large (15)	few (300)
GL3	500,000	150	small (5)	many (1,000)
GL4	500,000	150	large (15)	many (1,000)
GL5	1,000,000	175	small (5)	few (300)
GL6	1,000,000	175	large (15)	few (300)
GL7	1,000,000	175	small (5)	many (1,000)
GL8	1,000,000	175	large (15)	many (1,000)
RL1	987,486	133	small (1.0)	many (1,830)
RL1'	987,486	133	small (1.0)	many (1,830)
RL2	1,101,432	215	small (0.8)	many (3,196)
RL3	1,401,582	308	small (0.7)	many (5,833)
RL4	1,401,582	385	small (0.6)	many (5,833)
RVL1	2,171,792	50	large (17.3)	few (61)
RVL1'	2,171,792	50	large (17.3)	few (61)
RVL2	2,171,792	75	large (17.3)	few (61)
RVL3	2,171,792	100	large (16.6)	few (61)
RVL4	2,171,792	150	large (16.9)	few (61)
RVL5	2,171,792	200	large (16.9)	few (61)
RVL6	2,180,831	250	large (16.9)	few (108)
RVL7	2,180,831	295	large (16.3)	few (108)

GL1, RL1, RVL1 only in terms of constraints. The instances GS1', GM1', GL1', RL1', RVL1' are infeasible if all soft constraints are considered as hard constraints.

1.7.2 Experimental design

The matheuristic, the MBLP, and the MBLP′ are implemented in Python 3.7 and the Gurobi 8.1 solver is used. All computations are performed on an HP workstation with one Intel Xeon CPU with 2.20 GHz clock speed and 128 GB RAM. Even though the running time budget of the company is 30 minutes, we applied the exact approaches with a time limit of 10,000 seconds to obtain better reference values for evaluating the solutions of the matheuristic. For the matheuristic, we set parameter $k = 20$ for all instances. The matheuristic is further applied to selected instances with different values of k. In consultation with the company, the constants α, β, γ, and δ are each set to the maximum absolute expected profit of the corresponding instance. Setting α and δ to the maximum absolute expected profit ensures that the objective function value cannot be improved by having fewer assignments or contacts than prescribed by the bounds. This reflects the preference of the company to reach the prescribed bounds if possible. The company accepts a shortfall or an exceedance of the lower and upper bounds on the number of sales if the corresponding assignments have a small or a large expected profit, respectively. Setting β and γ to the maximum absolute expected profit is an adequate penalty for sales constraints from the perspective of the company. The reported running times of all approaches include the time to compute relevant sets and parameters, the time to set up and solve the optimization models with Gurobi, and the time used for the iterative algorithm of the matheuristic. The time used for importing and exporting data is excluded because it is equivalent for all three approaches.

1.7.3 Comparison of MBLP and MBLP′

First, we compare the performances of the MBLP and the MBLP′. Table 1.11 reports, for each instance and each formulation, the objective function value (OFV), the total penalty and in brackets the number of slack variables that take a positive value, the MipGap, the total number of constraints, and the total running time. The entry "lim" means that the time limit was reached. A dash (-) indicates that setting up the respective model resulted in an out-of-memory error. From Table 1.11, we can conclude that the MBLP′ has much fewer constraints than the MBLP for all instances. As a consequence, a larger number of feasible and also optimal solutions can be derived, and the running times are generally shorter. Figure 1.8 compares the running times of the MBLP and the MBLP′ using

Table 1.11: Results of the MBLP, the MBLP′ and the matheuristic for generated and real-world instances

	MBLP					MBLP′					Matheuristic					
ID	OFV [100k]	Penalty [100k] (#pos. slack.)	MipGap [%]	Constr. [1k]	CPU [sec]	OFV [100k]	Penalty [100k] (#pos. slack.)	MipGap [%]	Constr. [1k]	CPU [sec]	OFV [100k]	Penalty [100k] (#pos. slack.)	Sum slack var.	Constr. [1k]	CPU [sec]	Gap [%]
GS1	**1.5**	0.00 (0)	0.0	34	**3.4**	**1.5**	0.00 (0)	0.0	14	8.4	1.5	0.00 (0)	0.00	2	5.7	0.3
GS1'	**1.2**	0.27 (1)	0.0	34	**2.7**	**1.2**	0.27 (1)	0.0	14	8.2	1.2	0.27 (1)	47.00	2	4.3	0.4
GS2	**1.4**	0.00 (0)	0.0	172	8.1	**1.4**	0.00 (0)	0.0	36	11.5	1.4	0.00 (2)	0.03	4	**5.4**	1.7
GS3	**1.0**	0.00 (0)	0.0	41	**2.7**	**1.0**	0.00 (0)	0.0	17	9.0	1.0	0.00 (0)	0.00	5	7.9	0.3
GS4	**5.1**	0.00 (0)	0.0	180	11.4	**5.1**	0.00 (0)	0.0	36	14.7	5.0	0.00 (0)	0.00	7	**10.3**	1.6
GS5	**1.3**	0.00 (0)	0.0	36	**3.3**	**1.3**	0.00 (1)	0.0	20	8.9	1.3	0.00 (1)	0.05	2	4.8	0.2
GS6	**13.9**	0.00 (0)	0.0	815	37.6	**13.9**	0.00 (0)	0.0	83	26.7	13.5	0.01 (2)	1.00	4	**7.3**	2.5
GS7	**4.2**	0.00 (0)	0.0	72	**5.0**	**4.2**	0.00 (0)	0.0	29	10.3	4.2	0.01 (1)	1.00	4	8.9	0.5
GS8	**9.2**	0.00 (0)	0.0	736	39.9	**9.2**	0.00 (0)	0.0	83	29.8	8.9	0.00 (0)	0.00	8	**12.4**	2.9
GM1	**36.0**	0.00 (0)	0.0	1,154	95.0	**36.0**	0.00 (0)	0.0	275	78.0	35.5	0.00 (1)	0.05	18	**32.8**	1.4
GM1'	**32.8**	1.16 (1)	0.0	1,154	109.2	**32.8**	1.16 (1)	0.0	275	82.7	32.3	1.16 (2)	135.05	18	**32.9**	1.3
GM2	**88.4**	0.00 (0)	0.0	5,993	317.1	**88.4**	0.00 (0)	0.0	529	184.1	84.8	0.00 (1)	0.10	32	**49.8**	4.0
GM3	**25.1**	0.00 (0)	0.0	1,244	68.3	**25.1**	0.00 (0)	0.0	285	**52.3**	24.9	0.00 (1)	0.03	48	84.3	0.8
GM4	**30.1**	0.00 (0)	0.0	5,811	869.0	**30.1**	0.00 (0)	0.0	524	653.0	29.3	0.00 (0)	0.00	84	**114.7**	2.7
GM5	**47.6**	0.00 (0)	0.0	3,162	450.9	**47.6**	0.00 (0)	0.0	623	341.5	46.8	0.01 (2)	1.13	19	**39.0**	1.8
GM6	**147.4**	0.00 (0)	0.0	18,358	1,449.5	**147.4**	0.00 (0)	0.0	1,210	845.8	141.3	0.00 (1)	0.19	36	**65.6**	4.2
GM7	**71.4**	0.00 (0)	0.0	3,292	303.4	**71.4**	0.00 (0)	0.0	617	233.2	70.5	0.00 (2)	0.15	51	**97.8**	1.4
GM8	**130.0**	0.00 (0)	0.0	18,309	7,282.6	**130.0**	0.00 (0)	0.0	1,200	5,982.5	123.6	0.00 (0)	0.00	96	**150.8**	4.9
GL1	**248.8**	0.00 (0)	0.0	10,033	3,708.4	**248.8**	0.00 (0)	0.0	1,673	3,037.1	243.0	0.01 (2)	1.01	21	**58.3**	2.4
GL1'	**-389.7**	616.69 (2)	0.0	10,033	3,335.9	**-389.7**	616.69 (2)	0.0	1,673	2,696.2	-394.9	616.69 (2)	53,273.00	21	**57.2**	1.3
GL2	-172.2	295.48 (3)	402.4	69,790	lim	-172.2	295.48 (3)	402.1	3,533	lim	**256.1**	0.00 (0)	0.00	42	**115.1**	-248.7
GL3	**197.3**	0.00 (0)	0.0	10,137	1,371.5	**197.3**	0.00 (0)	0.0	1,641	1,034.9	194.0	0.00 (0)	0.00	67	**139.3**	1.7
GL4	58.2	31.69 (1)	630.9	67,492	lim	58.2	31.69 (1)	631.2	3,491	lim	**211.8**	0.01 (2)	1.30	139	**251.2**	-263.9
GL5	-229.9	323.26 (2)	216.3	25,676	lim	-229.9	323.26 (2)	216.3	3,496	lim	**202.0**	0.00 (1)	0.09	22	**89.7**	-187.8
GL6	-	- (-)	-	-	-	166.7	4.89 (1)	443.6	7,163	lim	**452.1**	0.00 (1)	0.53	43	**209.3**	-171.3
GL7	**475.3**	0.00 (0)	0.0	26,239	5,087.4	**475.3**	0.00 (0)	0.0	3,528	4,148.5	465.9	0.00 (1)	0.03	72	**185.4**	2.0
GL8	-	- (-)	-	-	-	-2,084.1	2,378.82 (3)	201.8	7,136	lim	**1,022.2**	0.00 (1)	0.01	143	**354.6**	-149.0
RL1	**109.8**	0.00 (0)	0.0	388	579.6	**109.8**	0.00 (0)	0.0	244	548.5	109.2	0.00 (0)	0.00	21	**46.5**	0.6
RL1'	**110.4**	1.83 (1)	0.0	388	623.4	**110.4**	1.83 (1)	0.0	244	593.6	109.7	1.83 (1)	415.00	21	**46.5**	0.6
RL2	**149.3**	0.00 (0)	0.0	755	2,570.6	**149.3**	0.00 (0)	0.0	538	2,352.9	148.7	0.00 (0)	0.00	39	**85.2**	0.4
RL3	**224.6**	0.00 (0)	0.0	2,079	4,937.4	**224.6**	0.00 (0)	0.0	1,012	4,097.6	223.0	0.00 (0)	0.00	85	**173.9**	0.7
RL4	**237.6**	0.00 (0)	0.0	3,200	5,523.6	**237.6**	0.00 (0)	0.0	1,197	5,044.9	236.0	0.00 (0)	0.00	106	**205.8**	0.7
RVL1	**292.6**	0.00 (0)	0.0	64,911	1,724.3	**292.6**	0.00 (0)	0.0	3,052	545.2	291.7	0.00 (0)	0.00	6	**100.2**	0.3
RVL1'	**247.8**	44.87 (1)	0.0	64,911	1,735.8	**247.8**	44.87 (1)	0.0	3,052	519.8	246.8	44.87 (1)	2,000.00	6	**99.8**	0.4
RVL2	-	- (-)	-	-	-	**378.6**	0.00 (0)	0.0	3,999	1,917.1	377.0	0.00 (0)	0.00	9	**144.9**	0.4
RVL3	-	- (-)	-	-	-	**537.1**	0.00 (0)	0.0	9,431	3,868.9	534.7	0.00 (0)	0.00	15	**182.3**	0.4
RVL4	-	- (-)	-	-	-	-	- (-)	-	-	-	**751.0**	0.00 (0)	0.00	28	**273.5**	-
RVL5	-	- (-)	-	-	-	-	- (-)	-	-	-	**975.9**	0.00 (0)	0.00	40	**365.2**	-
RVL6	-	- (-)	-	-	-	-	- (-)	-	-	-	**1,273.3**	0.00 (0)	0.00	84	**476.4**	-
RVL7	-	- (-)	-	-	-	-	- (-)	-	-	-	**1,501.8**	0.00 (0)	0.00	102	**554.0**	-

(-) Not available due to out-of-memory error

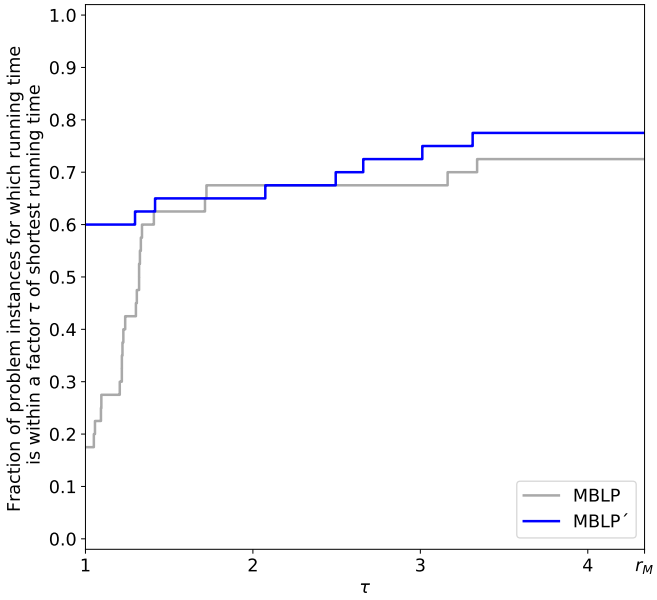

Figure 1.8: Performance profiles for the MBLP and the MBLP′ (cf. Dolan and Moré, 2002)

performance profiles (Dolan and Moré, 2002). Each curve corresponds to an approach and indicates for what fraction of problem instances the running time of the respective approach was within a factor τ of the shortest running time. The parameter r_M is set to the highest factor that occurs plus one; and if an approach cannot solve an instance to optimality within the time limit, the factor for the corresponding instance is set to r_M. From Figure 1.8 (at $\tau = 1$), we can see that the MBLP′ is the faster approach for 60% of the instances while the MBLP is faster for 17.5% of the instances. Note that the MBLP is only faster when solving small instances. The MBLP′ maintains a considerably larger fraction up to $\tau = 1.3$. Also for large values of τ (when the performance profiles become flat), we can see that the fraction of instances that can be solved to optimality within the time limit is higher for the MBLP′ than for the MBLP. To further investigate the substantial difference in the number of constraints of the MBLP and the MBLP′, we applied the MBLP′ with and without step 5) of the preprocessing technique. As explained in Section 1.6, step 5) removes the cliques from clique matrices that are a subset of another clique. Table 1.12 reports the total number of constraints to ensure conflict rules ($\sum$) for groups of instances for the MBLP, the MBLP′, and the MBLP′ without conducting step 5). The results clearly show that the preprocessing technique is effective and that step 5) is essential.

We also examined two alternatives to the use of soft constraints. First, we tested a

Table 1.12: Number of constraints to ensure the conflict rules

ID	MBLP	MBLP′	MBLP′ without step 5)
$\sum$ GS	2,039,866	250,808 (-88%)	1,528,941 (-25%)
$\sum$ GM	57,273,804	4,334,985 (-92%)	47,112,825 (-18%)
$\sum$ GL	569,555,040	27,113,838 (-95%)	422,135,617 (-26%)
$\sum$ (RL+RVL)	71,089,558	5,653,887 (-92%)	23,701,370 (-67%)

variant of the model MBLP′ in which all soft constraints are replaced by hard constraints. Of course, this variant was not able to devise solutions for the instances GS1', GM1', GL1', RL1', and RVL1'. For the other instances, the variant with hard constraints obtained identical or very similar results in terms of solution quality and running time as the variant with soft constraints.

Second, we tested a Lagrangian relaxation scheme (LRS). To obtain the LRS, we first formulated all soft constraints as hard constraints and deleted the penalty terms in the objective function. Next, we dualized the former soft constraints to obtain the Lagrangian subproblem. We iteratively solved the Lagrangian subproblem using the subgradient algorithm as described in Fisher (1981) and Fisher (1985). It turns out that for most instances, the LRS is inferior to the MBLP′ in terms of running time and solution quality. Only for some of the large instances, the LRS was able to devise better solutions than the MBLP′. However, these solutions are with one exception (GL5) still much worse than the solutions obtained by the matheuristic. For problem instance GL5, the LRS obtained a slightly better solution than the matheuristic (gap of 2.3%). The LRS also runs out of memory for the largest real-world problem instances. In our view, a main factor that negatively affects the performance of the LRS is that the Lagrangian subproblem in each iteration cannot be solved much faster than the original problem with hard or soft constraints because it still contains the large number of conflict constraints.

1.7.4 Performance of matheuristic

Next, we compare the performance of the matheuristic to the performance of the MBLP′. The right part of Table 1.11 reports the results of the matheuristic. The columns for the matheuristic are the same as for the MBLP′ except for the column sum slack variables, which states the sum of the slack variables, and the last column, which reports the gap between the OFV of the solution derived by the matheuristic and the OFV of the solution derived by the MBLP′. For each instance, we highlight the shortest running time and the highest OFV of all three approaches in bold. Note that some small positive penalties

are rounded down to zero in Table 1.11 (cf. e.g., instance GS2). First, we compare the matheuristic and the MBLP′ in terms of solution quality. Most of the solutions of the matheuristic obtained with $k = 20$ are near-optimal. For problem instance GM8, we investigate how changing the value of parameter k affects the gap to the optimal solution and the running time of the matheuristic. Figure 1.9 visualizes the gap to the optimal solution and the running time of the matheuristic for various values of k. We can see that the gap can be further reduced by increasing parameter k. Interestingly, the gap decreases faster than linearly, whereas the running time appears to increase linearly. Parameter k can therefore be used to control the trade-off between solution quality and speed. Overall, there are only a few slack variables that take positive values, and the resulting penalties for the matheuristic are minor. Exceptions are the instances GS1', GM1', GL1', RL1', RVL1' which are infeasible if all soft constraints are considered as hard constraints and thus, a (large) positive penalty value cannot be avoided.

Next, we compare the two approaches in terms of running time. The matheuristic is substantially faster than the MBLP′, especially for medium instances with a high eligibility fraction and for large and very large instances. Furthermore, the matheuristic is scalable to very large real-world instances. Figure 1.10 shows the running time for different steps of the matheuristic. From the bars, we can see that setting up the LP only plays a significant role if the instance has many eligibility patterns (cf. e.g., GL3–GL4, GL7–GL8, and RL1–RL4). In the first step of the matheuristic, customers who are eligible for the same activities are grouped together. Thus, instances with many eligibility patterns result in more groups and exhibit larger models, which explains the higher time consumption for setting up the LP for these instances. With increasing size of the RVL instances, primarily the running time for generating all sets and parameters increases (e.g., for deriving subsets of the activities that are associated with a specific constraint), whereas the rest of the steps require only a slightly longer running time. The preprocessing technique runs fast for all instances. Even for the largest instances, the matheuristic (with $k = 20$) always terminates in less than 10 minutes and is thus well within the running time budget prescribed by the company.

Next, we analyze the impact of the complexity parameters on the quality of the solutions obtained by the matheuristic. Therefore, we only consider the generated instances for which we systematically varied the complexity parameters. Generally, the average gaps increase with increasing eligibility fraction, as we can see from Table 1.13. While the eligibility fraction has an effect on the average gaps, the number of eligibility patterns has almost no effect. We can see from Table 1.11 that the gaps remain small even with increasing numbers of customers and activities in the instances. Overall, the matheuristic

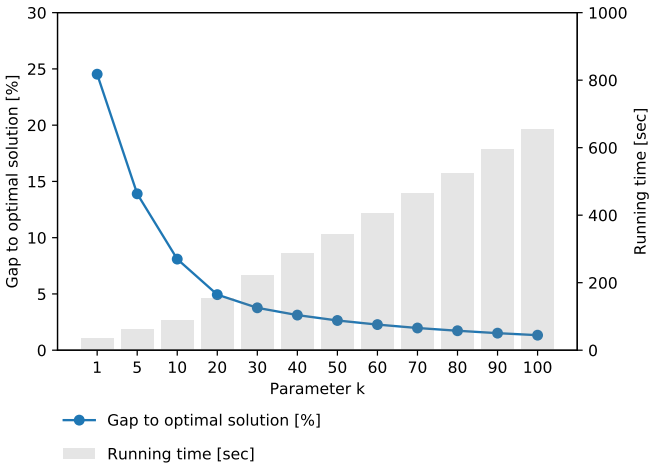

Figure 1.9: Instance GM8: gap to optimal solution vs. running time of the matheuristic for different values of k

Table 1.13: Average gaps of the matheuristic for generated instances by eligibility fraction and eligibility patterns. Only instances that are solved to optimality by the MBLP′ are considered.

		Eligibility patterns		
		few	many	aggregated
Eligibility fraction	small	1.1	1.1	1.1
	large	3.1	3.0	3.1
	aggregated	1.8	1.9	1.8

provides high-quality solutions in a shorter running time than the MBLP and the MBLP′.

Finally, we performed two experiments to assess the effect of two key components of the matheuristic on running time and solution quality. In the first experiment, we analyze the impact of step 5) of the preprocessing technique and in the second experiment, we analyze the impact of the new modeling technique used to consider the conflict constraints in the LP.

Table 1.14 summarizes the results of the first experiment. It shows the total number of constraints to ensure conflict rules (# conflict const. in LP) for groups of instances ($\sum$) for the matheuristic and the matheuristic without conducting step 5) of the preprocessing technique. For the matheuristic without step 5), the increase in percent of the number of constraints to ensure conflict rules in the LP is stated in brackets (increase). Moreover, Table 1.14 states the total running time of both approaches (CPU). We can see that applying step 5) removes a substantial number of conflict constraints in the LP, and that setting up a smaller model leads to a lower total running time for all instance groups.

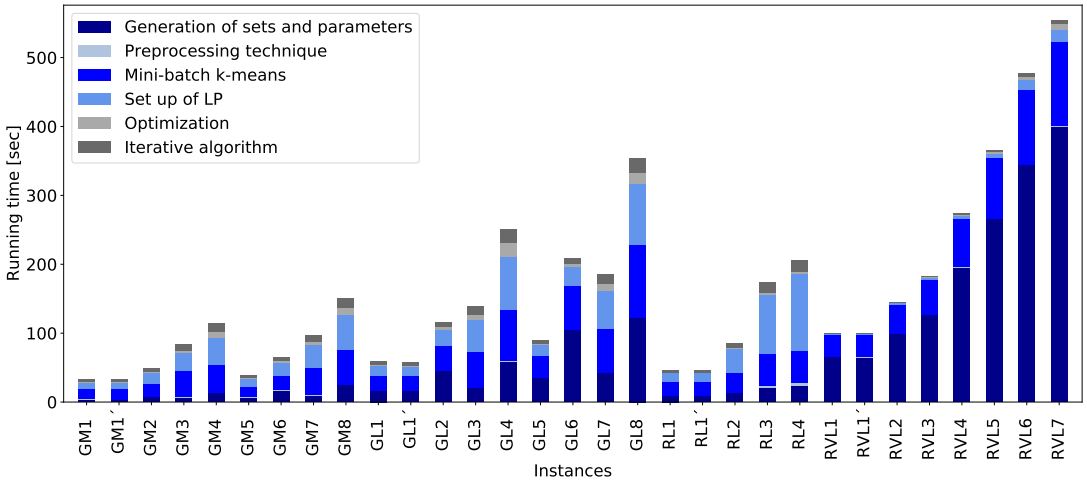

Figure 1.10: Running times for the different matheuristic steps

Table 1.14: Number of constraints to ensure the conflict rules in matheuristic

	Matheuristic		Matheuristic without step 5)	
ID	#conflict const. in LP	CPU [s]	#conflict const. in LP (increase)	CPU [s]
$\sum$ GS	24,910	**67.1**	138,030 (+454.1%)	73.7
$\sum$ GM	307,346	**667.6**	3,285,126 (+968.9%)	837.9
$\sum$ GL	458,967	**1,460.0**	7,404,226 (+1,513.2%)	1,912.5
$\sum$ (RL+RVL)	389,873	**2,754.1**	1,570,423 (+302.8%)	2,905.1

Note that the solution quality is not affected by step 5) of the preprocessing technique.

In the second experiment, we compare the proposed matheuristic to a benchmark version of the matheuristic that does not use the new modeling technique to incorporate the conflict constraints in the linear program. Instead of using the new modeling technique, the benchmark version incorporates the conflict constraints in the linear program by formulating constraints (9) of the MBLP for groups of customers. The resulting linear program $\overline{\text{LP}}$ reads as follows:

$$(\overline{\text{LP}}) \begin{cases} \text{Max. (15)} \\ \text{s.t.} \quad \text{(11)–(13), (16)–(22), (24), (25)} \\ \qquad x_{gj_1} + x_{gj_2} \leq o_g \qquad (g \in G;\ (j_1, j_2) \in T:\ j_1, j_2 \in J_g) \qquad (27) \end{cases}$$

Table 1.15 summarizes the results of the second experiment. It reports the objective function value (OFV), the penalty, the number of assignments (#agmts) in the LP solution (or in the $\overline{\text{LP}}$ solution), the number of assignments that are conducted in the iterative

algorithm (#agmts (it. alg.)), and the total running time (CPU). The results in Table 1.15 demonstrate the advantages of using the new modeling technique. The new modeling technique considers the customer-specific constraints very effectively already in the group-level model such that almost all assignments in the LP solution can be conducted by the iterative algorithm. In contrast, the $\overline{\text{LP}}$ solution has many assignments that cannot be conducted by the iterative algorithm because they would violate customer-specific conflict constraints. When a large fraction of assignments cannot be conducted by the iterative algorithm, some bounds of minimum assignment and sales constraints that were satisfied in the group-level model are violated after applying the iterative algorithm. This explains the large penalties and hence lower objective function values of the solutions of the matheuristic which uses model $\overline{\text{LP}}$. The new modeling technique not only improves the solution quality but also reduces the running time because the group-level model has fewer constraints and a smaller number of assignments needs to be processed by the iterative algorithm.

1.8 Conclusion

In this study, we introduced a real-world planning problem of a telecommunications company. The planning problem consists of assigning existing customers to direct marketing activities subject to various business constraints, such as budget and sales constraints, and various customer-specific constraints. The customer-specific constraints ensure, for example, that individual customers are generally not assigned to the activities too often and that the customers are not assigned to activities that are subject to a conflict. Such a conflict may exist, for example, between two activities that are scheduled within the same week. Existing approaches that deal with customer assignment in direct marketing do not consider such customer-specific constraints and are thus not applicable to the planning problem at hand. We developed a matheuristic that first solves an optimization problem for groups of customers and then iteratively assigns individual customers to the activities based on the solution to the group-level problem. New modeling techniques and a preprocessing technique are introduced to consider customer-specific constraints already in the group-level model. In a computational analysis, we demonstrated the effectiveness of the preprocessing technique and the problem decomposition strategy of the matheuristic based on a test set that includes generated and real-world instances. The proposed preprocessing technique is able to reduce the number of constraints in the models by up to 95%. Even when the number of groups is relatively small, the average gap of the solutions derived by the matheuristic to the optimal solutions of the generated instances

is only 1.8%. Increasing the number of groups further reduces the gap while prolonging the running time only slightly. The matheuristic is currently in use at the company and has lead to an overall improvement of its key performance indicators. The company estimates based on a proof of benefit conducted on a selected campaign that the use of the matheuristic increased the number of sales by 90%, which improved the profitability of this campaign by around 300%.

In future research, we will extend the planning problem to include multi-stage campaigns. In such campaigns, customers can only be assigned to activities of a non-initial stage when they have been assigned to an activity of each previous stage. A promising direction for future research is the development of strategies for grouping customers with different eligibility patterns, as pointed out in Section 1.5.1. Moreover, it would be interesting to adapt the preprocessing technique and the decomposition strategy to related planning problems such as the bin packing problem with conflict constraints. Finally, another direction for future research is to incorporate the conflict constraints with branching rules as done in the exact solution approaches of Şuvak et al. (2020) for the maximum flow problem with conflict constraints and of Şuvak et al. (2021) for the minimum cost flow problem with conflict constraints.

Acknowledgement

We would like to thank the company for their support and for the excellent collaboration.

Table 1.15: Effectiveness of new modeling technique to incorporate conflict constraints in group-level model

	Matheuristic					Matheuristic without new modeling technique				
ID	OFV [100k]	Penalty [100k]	#agmts (LP)	#agmts (it. alg.)	CPU [s]	OFV [100k]	Penalty [100k]	#agmts ($\overline{\text{LP}}$)	#agmts (it. alg.)	CPU [s]
GS1	**1.5**	0.0	8,331	8,331	5.7	1.0	0.4	8,600	7,946	**5.4**
GS1'	**1.2**	0.3	8,331	8,331	**4.3**	0.7	0.7	8,600	7,946	6.3
GS2	**1.4**	0.0	17,079	17,079	**5.4**	-0.1	1.3	20,020	15,722	7.7
GS3	**1.0**	0.0	9,826	9,826	**7.9**	0.9	0.1	9,943	9,263	9.3
GS4	**5.0**	0.0	17,594	17,594	**10.3**	3.2	1.0	19,193	15,786	13.2
GS5	**1.3**	0.0	16,698	16,698	**4.8**	1.3	0.0	18,295	16,014	5.5
GS6	**13.5**	0.0	40,975	40,975	**7.3**	1.2	10.2	51,067	34,559	10.6
GS7	**4.2**	0.0	20,897	20,897	**8.9**	3.1	0.7	23,304	19,530	10.6
GS8	**8.9**	0.0	38,903	38,902	**12.4**	-3.7	10.8	48,212	33,501	18.9
GM1	**35.5**	0.0	149,790	149,790	**32.8**	3.3	27.5	184,503	135,298	45.7
GM1'	**32.3**	1.2	143,902	143,902	**32.9**	-46.2	76.4	174,792	133,806	46.7
GM2	**84.8**	0.0	220,773	220,773	**49.8**	31.1	35.3	279,628	175,282	81.2
GM3	**24.9**	0.0	151,642	151,639	**84.3**	-18.0	38.1	177,748	130,791	118.8
GM4	**29.3**	0.0	215,672	215,672	**114.7**	-75.5	98.0	257,281	174,272	216.4
GM5	**46.8**	0.0	310,369	310,350	**39.0**	14.5	25.6	390,896	274,002	59.4
GM6	**141.3**	0.0	448,297	448,297	**65.6**	71.2	41.4	573,383	365,234	120.7
GM7	**70.5**	0.0	320,507	320,507	**97.8**	-22.9	81.0	401,804	279,353	155.2
GM8	**123.6**	0.0	461,223	461,223	**150.8**	49.8	42.0	564,119	345,302	308.6
GL1	**243.0**	0.0	834,008	834,008	**58.3**	28.3	186.5	1,030,637	760,945	83.1
GL1'	**-394.9**	616.7	838,189	838,189	**57.2**	-492.7	694.0	988,698	787,555	81.6
GL2	**256.1**	0.0	1,149,637	1,149,637	**115.1**	190.7	15.4	1,468,184	931,097	179.2
GL3	**194.0**	0.0	787,395	787,395	**139.3**	146.3	24.0	951,477	712,955	244.0
GL4	**211.8**	0.0	1,193,477	1,193,477	**251.2**	-326.5	499.1	1,446,367	936,972	499.2
GL5	**202.0**	0.0	1,637,333	1,637,333	**89.7**	124.7	49.5	2,086,706	1,461,946	125.0
GL6	**452.1**	0.0	2,418,241	2,418,241	**209.3**	312.0	67.9	2,901,633	1,956,749	306.9
GL7	**465.9**	0.0	1,611,220	1,611,220	**185.4**	-404.3	801.3	2,003,448	1,427,594	314.2
GL8	**1,022.2**	0.0	2,378,114	2,378,114	**354.6**	74.4	762.1	2,942,362	1,861,592	830.3
RL1	**109.2**	0.0	231,224	231,224	**46.5**	109.0	0.0	231,047	230,778	95.8
RL1'	**109.7**	1.8	240,704	240,704	**46.5**	109.0	2.3	240,494	240,008	95.3
RL2	**148.7**	0.0	369,377	369,377	**85.2**	147.0	0.0	368,244	365,422	232.9
RL3	**223.0**	0.0	534,595	534,595	**173.9**	213.5	1.6	545,604	518,135	614.8
RL4	**236.0**	0.0	596,515	596,515	**205.8**	217.3	4.1	594,679	565,202	814.5
RVL1	**291.7**	0.0	186,499	186,499	**100.2**	236.4	0.0	186,125	159,361	101.8
RVL1'	**246.8**	44.9	187,086	187,086	**99.8**	191.5	44.9	186,712	159,948	102.7
RVL2	**377.0**	0.0	257,332	257,309	**144.9**	288.6	0.0	279,060	195,974	148.6
RVL3	**534.7**	0.0	350,901	350,751	**182.3**	403.7	0.0	372,490	274,505	187.9
RVL4	**751.0**	0.0	510,249	509,998	**273.5**	554.1	0.0	554,051	380,332	282.8
RVL5	**975.9**	0.0	658,216	657,843	**365.2**	742.1	0.0	723,998	497,519	377.2
RVL6	**1,273.3**	0.0	823,361	822,628	**476.4**	959.8	6.9	886,967	612,202	506.0
RVL7	**1,501.8**	0.0	941,824	941,284	**554.0**	1,119.8	0.0	1,004,740	676,107	592.6

Bibliography

Bettinelli, A., Cacchiani, V., Malaguti, E., 2017. A branch-and-bound algorithm for the knapsack problem with conflict graph. INFORMS Journal on Computing 29, 457–473.

Bhaskar, T., Sundararajan, R., Krishnan, P.G., 2009. A fuzzy mathematical programming approach for cross-sell optimization in retail banking. Journal of the Operational Research Society 60, 717–727.

Bigler, T., Baumann, P., Kammermann, M., 2019. Optimizing customer assignments to direct marketing activities: A binary linear programming formulation, in: Wang, M., Li, J., Tsung, F., Yeung, A. (Eds.), Proceedings of the 2019 IEEE International Conference on Industrial Engineering and Engineering Management, Macau. pp. 601–605.

Bron, C., Kerbosch, J., 1973. Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM 16, 575–577.

Cetin, F., Alabas-Uslu, C., 2017. Heuristic solution to the product targeting problem based on mathematical programming. International Journal of Production Research 55, 3–17.

Coelho, V.N., Oliveira, T.A., Coelho, I.M., Coelho, B.N., Fleming, P.J., Guimarães, F.G., Ramalhinho, H., Souza, M.J., Talbi, E.G., Lust, T., 2017. Generic pareto local search metaheuristic for optimization of targeted offers in a bi-objective direct marketing campaign. Computers & Operations Research 78, 578–587.

Cohen, M.D., 2004. Exploiting response models — optimizing cross-sell and up-sell opportunities in banking. Information Systems 29, 327–341.

Coniglio, S., Furini, F., San Segundo, P., 2021. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. European Journal of Operational Research 289, 435–455.

Darmann, A., Pferschy, U., Schauer, J., Woeginger, G.J., 2011. Paths, trees and matchings under disjunctive constraints. Discrete Applied Mathematics 159, 1726–1735.

Delanote, S., Leus, R., Nobibon, F.T., 2013. Optimization of the annual planning of targeted offers in direct marketing. Journal of the Operational Research Society 64, 1770–1779.

Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Mathematical Programming 91, 201–213.

Elhedhli, S., Li, L., Gzara, M., Naoum-Sawaya, J., 2011. A branch-and-price algorithm for the bin packing problem with conflicts. INFORMS Journal on Computing 23, 404–415.

Fisher, M.L., 1981. The lagrangian relaxation method for solving integer programming problems. Management Science 27, 1–18.

Fisher, M.L., 1985. An applications oriented guide to lagrangian relaxation. Interfaces 15, 10–21.

Garey, M.R., Johnson, D.S., 2002. Computers and Intractability Vol. 29. WH Freeman, New York.

Hagberg, A.A., Schult, D.A., Swart, P.J., 2008. Exploring network structure, dynamics, and function using NetworkX, in: Varoquaux, G., Vaught, T., Millman, J. (Eds.), Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena. pp. 11–15.

Lessmann, S., Haupt, J., Coussement, K., De Bock, K.W., 2021. Targeting customers for profit: An ensemble learning framework to support marketing decision-making. Information Sciences 557, 286–301.

Ma, S., Fildes, R., 2017. A retail store SKU promotions optimization model for category multi-period profit maximization. European Journal of Operational Research 260, 680–692.

Ma, S., Hou, L., Yao, W., Lee, B., 2016. A nonhomogeneous hidden Markov model of response dynamics and mailing optimization in direct marketing. European Journal of Operational Research 253, 514–523.

Miguéis, V.L., Camanho, A.S., Borges, J., 2017. Predicting direct marketing response in banking: comparison of class imbalance methods. Service Business 11, 831–849.

Nair, S.K., Tarasewich, P., 2003. A model and solution method for multi-period sales promotion design. European Journal of Operational Research 150, 672–687.

Nobibon, F.T., Leus, R., Spieksma, F.C., 2011. Optimization models for targeted offers in direct marketing: Exact and heuristic algorithms. European Journal of Operational Research 210, 670–683.

Oliveira, T.A., Coelho, V.N., Souza, M.J., Boava, D.L.T., Boava, F., Coelho, I.M., Coelho, B.N., 2015. A hybrid variable neighborhood search algorithm for targeted offers in direct marketing. Electronic Notes in Discrete Mathematics 47, 205–212.

Öncan, T., Şuvak, Z., Akyüz, M.H., Altınel, İ.K., 2019. Assignment problem with conflicts. Computers & Operations Research 111, 214–229.

Sadykov, R., Vanderbeck, F., 2013. Bin packing with conflicts: a generic branch-and-price algorithm. INFORMS Journal on Computing 25, 244–255.

Sculley, D., 2010 Web-scale k-means clustering, in: Rappa, M., Jones, P., Freire, J., Chakrabarti, S. (Eds.), Proceedings of the 19th International Conference on World Wide Web, Raleigh. pp. 1177–1178.

Sun, M., 2002. The transportation problem with exclusionary side constraints and two branch-and-bound algorithms. European Journal of Operational Research 140, 629–647.

Sundararajan, R., Bhaskar, T., Sarkar, A., Dasaratha, S., Bal, D., Marasanapalle, J.K., Zmudzka, B., Bak, K., 2011. Marketing optimization in retail banking. Interfaces 41, 485–505.

Şuvak, Z., Altınel, İ.K., Aras, N., 2020. Exact solution algorithms for the maximum flow problem with additional conflict constraints. European Journal of Operational Research 287, 410–437.

Şuvak, Z., Altınel, İ.K., Aras, N., 2021. Minimum cost flow problem with conflicts. Networks 78, 421–442.

Walteros, J.L., Buchanan, A., 2020. Why is maximum clique often easy in practice? Operations Research 68, 1866–1895.

Paper II

MIP-based approaches for multi-site project scheduling[2]

Tamara Bigler Mario Gnägi Norbert Trautmann

Department of Business Administration
University of Bern

Contents

[2]Published in Annals of Operations Research, to appear (DOI:10.1007/s10479-022-05109-0).

Abstract

The execution of a project is often distributed among multiple sites. The planning of such a project includes selecting a specific site for the execution of each of the project's activities and allocating the available resource units to the execution of these activities over time. While some resource units are available at a certain site only, others can be moved across sites. Given the spatial distance between sites, transportation times arise if a resource unit must be transported from one site to another or if the output of an activity must be transported to another site. This planning problem has been introduced in recent literature as the multi-site resource-constrained project scheduling problem. We present a continuous-time model and devise a matheuristic for this planning problem. The continuous-time model uses, among others, binary variables to impose a sequence between activities assigned to the same resource units. In the matheuristic, the binary restrictions on these variables are initially relaxed and iteratively restored for the subset of activities scheduled in the current iteration. We compare the performance of the continuous-time model and the matheuristic to the performance of a discrete-time model and several metaheuristics from the literature using two sets of test instances from the literature. Both the continuous-time model and the matheuristic derive on average superior solutions in shorter average running times than the reference approaches.

2.1 Introduction

Recently, Laurent et al. (2017) introduced the multi-site resource-constrained project scheduling problem, or short multi-site RCPSP, which allows to consider the execution of a project's activities at several sites and the sharing of resources among these sites. This planning problem is motivated by a real-world application from health care management, where medical examinations are conducted in different hospitals and the required medical staff is shared between these hospitals. According to Laurent et al. (2017), this pooling of medical examinations and staff increases the hospitals' total productivity even though additional transportation times occur due to the spatial distance between the hospitals. Another sample application are R&D projects whose research activities may be carried out in several laboratories that share some of their resources. In this application, mobile

resource units, e.g., research staff or equipment, or the output of some research activities, e.g., laboratory samples or prototypes, must be transported from one laboratory to another, resulting in transportation times that must be considered. The multi-site RCPSP is an extension of the widely studied single-site RCPSP, which represents an NP-hard optimization problem. Consequently, the multi-site RCPSP is also NP-hard.

The multi-site RCPSP—similar to the single-site RCPSP—consists of allocating individual units of some renewable resources over time to the execution of the activities of a project such that the project duration is minimized while taking into account a prescribed set of completion-start precedence relationships between pairs of activities. In addition, in the single-site RCPSP, it is assumed that all activities are executed at the same site and, consequently, that all resources are located at this site. In the multi-site RCPSP, in contrast, a specific site for the execution of each activity must be selected. Moreover, the spatial distance between the sites gives rise to two different types of transportation times that must be considered in the project schedule. First, during the transport of a resource unit between the sites, the resource unit cannot be allocated to the execution of an activity. Second, if two activities that are interrelated by a precedence relationship are executed at different sites, then a minimum time lag between the completion of the first activity and the start of the second activity must be taken into account, which corresponds to the time required to transport the first activity's output between the respective sites. Hereafter, we refer to the first type of transport as resource transport and to the second type of transport as output transport.

To the best of our knowledge, the only solution approaches to the multi-site RCPSP that are documented in the literature are the formulation as a binary linear program and the four different metaheuristics proposed by Laurent et al. (2017). The linear programming formulation belongs to the class of discrete-time models; i.e., the planning horizon is divided into a set of equally long periods, and it is assumed that an activity can be started at the beginning of such a period only. In an experimental analysis performed by Laurent et al. (2017) with CPLEX using a set of 192 instances, where the number of activities was varied between 5 and 30, it turned out that within a prescribed time limit of 3,600 seconds of running time, none of the instances comprising 30 activities could be solved to optimality. The four metaheuristics are based on a representation of feasible solutions by an activity list and a site list and apply the search strategies local search, simulated annealing and iterated local search. In another experimental analysis performed by Laurent et al. (2017) with instances comprising 30 to 120 activities, it turned out that the iterated local search and the simulated annealing metaheuristics performed best.

The contribution of this paper is twofold: first, we provide a continuous-time model

for the multi-site RCPSP that is applicable to challenging instances comprising up to 30 activities such that it can be used to evaluate the performance of heuristic methods; and second, we devise a novel matheuristic for the multi-site RCPSP that follows an iterative, relax-optimize-and-fix strategy and applies a relaxation of the novel continuous-time model in each iteration. More specifically, the novel model, which we elaborate starting from the single-site RCPSP formulation presented in Rihm and Trautmann (2017), is formulated as a mixed-binary linear programming (MBLP) model and employs continuous start-time variables and binary site-selection, resource-assignment and sequencing variables. In an experimental performance analysis, we apply the novel continuous-time model and the discrete-time model proposed by Laurent et al. (2017) to a set of 960 instances comprising 30 activities and 2 or 3 sites that were generated by Laurent et al. (2017). Feasible solutions are devised for all instances with the novel continuous-time model, and a large number of these instances can even be solved to optimality. Moreover, for a considerable number of instances, using the novel continuous-time model yields a feasible solution that has a better objective function value than the best solution devised by the discrete-time model presented in Laurent et al. (2017) in a shorter average running time. Based on this novel continuous-time model, we propose a matheuristic for larger instances. In this matheuristic, subsets of the activities are iteratively scheduled using a relaxation of the model. All activities of the instance are considered in this relaxation. During the solution process, however, several activities may overlap with each other, i.e., use the same resource unit at the same time. In each iteration, the activities in the respective subset are sequenced among themselves and among the already-scheduled activities, until after the last iteration, no overlaps remain. The matheuristic obtains good feasible solutions for instances comprising 30 activities with a lower average gap to the critical-path-based lower bound than the continuous-time model. For the instances comprising 60 activities, it outperforms the metaheuristics of Laurent et al. (2017) on all considered performance metrics.

The remainder of this paper is organized as follows. In Section 2.2, we illustrate the multi-site RCPSP by an example and provide an overview of related project scheduling problems. In Sections 2.3 and 2.4, we present the novel continuous-time model and the matheuristic. In Section 2.5, we report our computational results. In Section 2.6, we provide some conclusions and provide an outlook for future research.

2.2 Planning problem

In this section, we introduce some basic notation (Section 2.2.1), illustrate the planning problem by means of an example project (Section 2.2.2), and provide an overview of related project scheduling problems with transportation times discussed in the literature (Section 2.2.3).

2.2.1 Basic notation

We consider a project that consists of n real activities. These real activities form the set V. In addition, we introduce two fictitious activities, 0 and $n+1$, representing the start and end of the project, respectively; both of these activities are assumed to have a duration of zero and to require no resources. Each real activity $i \in V$ has a prescribed duration p_i and must not be pre-empted after it has been started. Furthermore, given is a set E containing the pairs of activities between which there is a completion-start precedence relationship. For the execution of the activities, a set R of different renewable resource types and a set L of alternative sites are given. For each resource type $k \in R$, we denote the available number of units as R_k and the required number of units for the execution of activity $i \in V$ as $r_{ik} \leq R_k$. Moreover, each available unit $u \in \{1, \ldots, R_k\}$ of resource type $k \in R$ is either mobile (i.e., $M_{ku} = 1$) or nonmobile (i.e., $M_{ku} = 0$). All mobile resource units are assumed to be initially located at the site at which their first assigned real activity is executed, i.e., the mobile resource units do not have to be moved before the start of the first real activity assigned to them. Moreover, the mobile resource units are assumed to remain at the site at which the last real activity assigned to them is executed, i.e., the mobile resource units do not have to be moved after the completion of the last real activity assigned to them. For each nonmobile resource unit, there is a site loc_{ku} at which the resource unit is permanently located. Finally, $\delta_{ll'}$ denotes the transportation time between sites l and $l' \in L \times L$.

2.2.2 Illustration of the planning problem

We illustrate the planning problem by means of an example project that comprises seven real activities, i.e., $V = \{1, \ldots, 7\}$. For the execution of these activities, there are two sites, A and B, i.e., $L = \{\mathrm{A}, \mathrm{B}\}$, between which we assume a transportation time of one time unit in both directions, i.e., $\delta_{AB} = \delta_{BA} = 1$. Moreover, two different resource types are required to execute the activities, i.e., $R = \{1, 2\}$, with two resource units of both resource types available, i.e., $R_1 = 2$ and $R_2 = 2$. Both resource units of resource

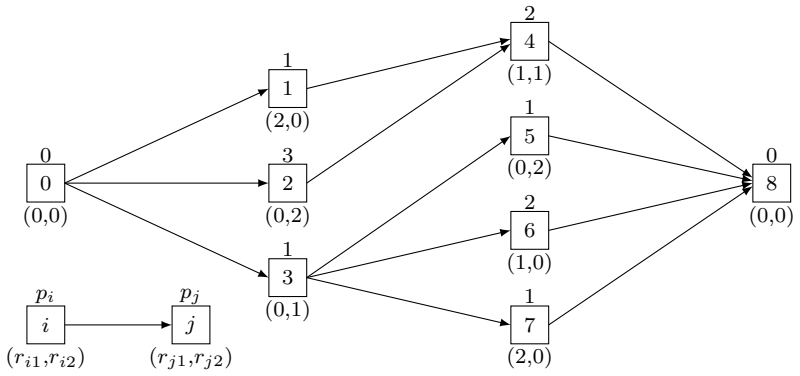

Figure 2.1: Activity-on-node network for the example project

type $k = 1$ are nonmobile, i.e., $M_{11} = M_{12} = 0$, and permanently located at site A, i.e., $loc_{11} = loc_{12} = \text{A}$. The first unit of resource type $k = 2$ is also nonmobile, i.e., $M_{21} = 0$, and permanently located at site B, i.e., $loc_{21} = \text{B}$, while the second unit is mobile, i.e., $M_{22} = 1$. Figure 2.1 shows an activity-on-node network, which depicts the activities as nodes and the precedence relationships as directed edges between the nodes, for the example project. The durations and resource requirements of the activities are indicated above and below the nodes of the network, respectively.

An optimal solution for the example project with a minimal makespan of eight time units is shown in Figure 2.2. Each line represents a resource unit u of a resource type k, and the rectangles represent the activities. Each real activity is assigned to at least one resource unit and exactly one site. Activities $1, 4, 6$ and 7 are executed at site A, while activities $2, 3$ and 5 are executed at site B. Resource transport, e.g., between activities 5 and 4, is indicated by a rectangle with a cross. These two activities are both assigned to the resource unit $u = 2$ of resource type $k = 2$ and take place at different sites. Thus, the commonly used resource unit must be moved from site B to site A before activity 4 can start. Output transports, e.g., between activities 3 and 7, are indicated with arrows. These two activities are precedence related and take place at different sites. Thus, the output of activity 3 must be moved from site B to site A before activity 7 can start.

2.2.3 Related project scheduling problems with transportation times

In addition to the work of Laurent et al. (2017), which introduces and addresses the multi-site RCPSP for the first time, there are other papers in the literature that discuss related project scheduling problems. In the following, we outline these related project

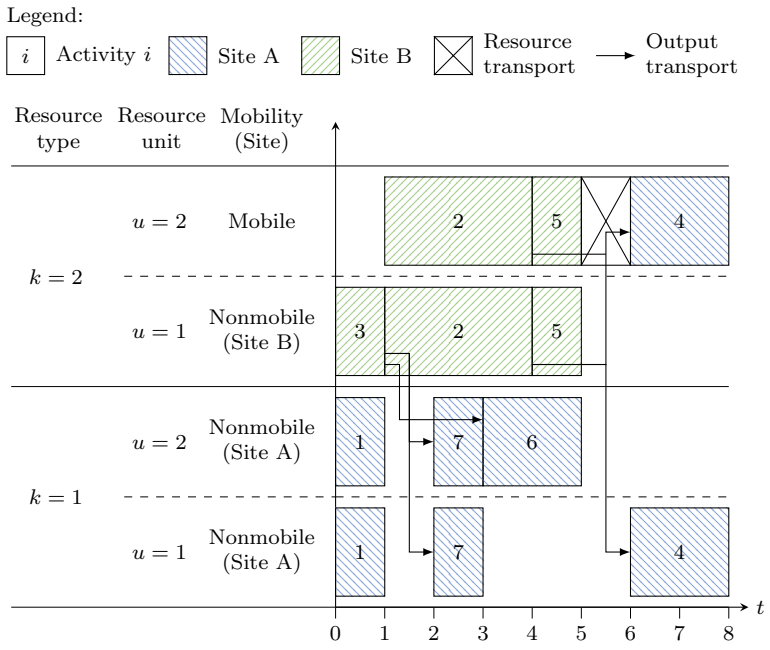

Figure 2.2: An optimal solution for the example project

planning problems and point out the differences to the problem considered in this paper.

A well-known scheduling problem that has similar characteristics as the multi-site RCPSP is the multi-mode RCPSP (cf., e.g., Gnägi et al., 2019). In the multi-mode RCPSP, the project activities can be executed in alternative modes that differ in terms of the activities' durations and resource requirements. Selecting an execution mode for an activity can be interpreted as assigning the activity to a specific site. The selection of an execution mode for an activity, however, affects only the duration of this specific activity individually. Therefore, output transport cannot be addressed since such transport always includes two activities. Resource transport cannot be addressed, either, since the movement of mobile resource units between different sites cannot be modeled based on mode selection. Several extensions of the multi-mode RCPSP that explicitly consider transportation times (also called time lags, setup times or transfer times) are discussed in the literature. In the multi-mode RCPSP with mode-dependent time lags, which has been discussed by Sabzehparvar and Seyed-Hosseini (2008), the length of the time lags depends on the selected modes of two activities. Therefore, when the mode selection is interpreted as a site selection, output transport, as in the multi-site RCPSP, may be addressed. However, the sharing of mobile resource units among different sites cannot be addressed. The multi-mode RCPSP with schedule-dependent setup times, considered by Mika et al. (2004), explicitly involves the assignment of activities to locations

and the movement of semifinished products between the locations of precendence-related activities. The required time for this movement is interpreted as a setup time for the subsequent activity. However, also in this planning problem, the locations of the resource units are assumed to be fixed, and thus, no sharing of mobile resource units among different locations is addressed. Another extension, namely, the multi-mode RCPSP with sequence-dependent transfer times, is considered by Kadri and Boctor (2014). Transfer times occur when resource units are moved between several locations, and the duration of the transfers depends on the locations at which the involved activities are executed. Unlike in the multi-site RCPSP, however, in this case, the locations for the execution of the activities are assumed to be given.

A less related stream of literature, initiated by Krüger and Scholl (2009, 2010), deals with the RCPSP with transfer times in the context of single- and multi-project scheduling. In these problem settings, transfer times occur when resource units are transferred from one activity to another, while the duration of the transfer depends on the involved activities as well as the resource type. These transfer times may occur within the same project but also among multiple projects, but the belonging of the activities to the projects is considered as given. Therefore, the multi-project context cannot be exploited to model the selection of sites for the execution of the activities. The identical planning problem in the context of single-project scheduling is considered by Poppenborg and Knust (2016), Kadri and Boctor (2018) and Liu et al. (2021). Liu et al. (2021) focus on the special case where only resources with one available unit are considered.

In sum, some of the abovementioned papers address isolated parts of the problem discussed in this paper. Most of them do not explicitly involve the selection of sites for the execution of the activities and the transport of semifinished products between the sites of precedence-related activities, and none of them accounts for the sharing of mobile resource units between different sites.

2.3 Continuous-time MBLP model

In this section, we present the continuous-time MBLP model for the multi-site RCPSP; a preliminary version of this model is presented in Gnägi and Trautmann (2019). In Section 2.3.1, we illustrate the different types of variables by means of the example project introduced in Section 2.2. In Section 2.3.2, we present the formulation of the objective and the constraints.

Table 2.1: Variable descriptions

Variable	Description
$*$ S_i	Starting time of activity i
$*$ s_{il}	$\begin{cases} = 1, \text{ if activity } i \text{ is executed at site } l \\ = 0, \text{ otherwise} \end{cases}$
$*$ r_{ik}^u	$\begin{cases} = 1, \text{ if activity } i \text{ is assigned to unit } u \text{ of resource type } k \\ = 0, \text{ otherwise} \end{cases}$
$*$ y_{ij}	$\begin{cases} = 1, \text{ if activity } i \text{ must be completed before the start of activity } j \\ = 0, \text{ otherwise} \end{cases}$

2.3.1 Types of variables

The model employs the four types of variables listed in Table 2.1. The continuous start-time variables S_i ($i \in V \cup \{0, n+1\}$) indicate when to start the activities. The project is assumed to start at time point zero, i.e., $S_0 := 0$, and consequently, the starting time of activity $n+1$ represents the project duration. The binary site-selection variables s_{il} ($i \in V;\ l \in L$) indicate at which site the activities are executed. The binary resource-assignment variables r_{ik}^u ($i \in V;\ k \in R;\ u \in \{1, \ldots, R_k\} :\ r_{ik} > 0$) indicate to which resource units the activities are assigned. Finally, the binary sequencing variables y_{ij} ($i, j \in V \times V : i \neq j,\ (i,j) \notin TE$) indicate the sequence in which two activities i and j must be executed. Note that the sequencing variables are only defined for all pairs of activities (i,j) with $i \neq j$ which can be executed simultaneously with respect to the precedence relationships, i.e., $(i,j) \notin TE$. If two of these activities, i.e., activities i and j, use at least one common resource unit of the same resource type, then a sequencing between the two activities must be ensured, i.e., either y_{ij} or y_{ji} will take the value one. Figure 2.3 illustrates the four types of variables by means of the example project introduced in Section 2.2. For the binary variables, only those variables with a positive value in the given optimal solution are displayed.

2.3.2 Formulation of objective and constraints

The objective is to minimize the duration of the project:

$$\text{Min. } S_{n+1}$$

Constraints (2.1) account for the prescribed precedence relationships among the real

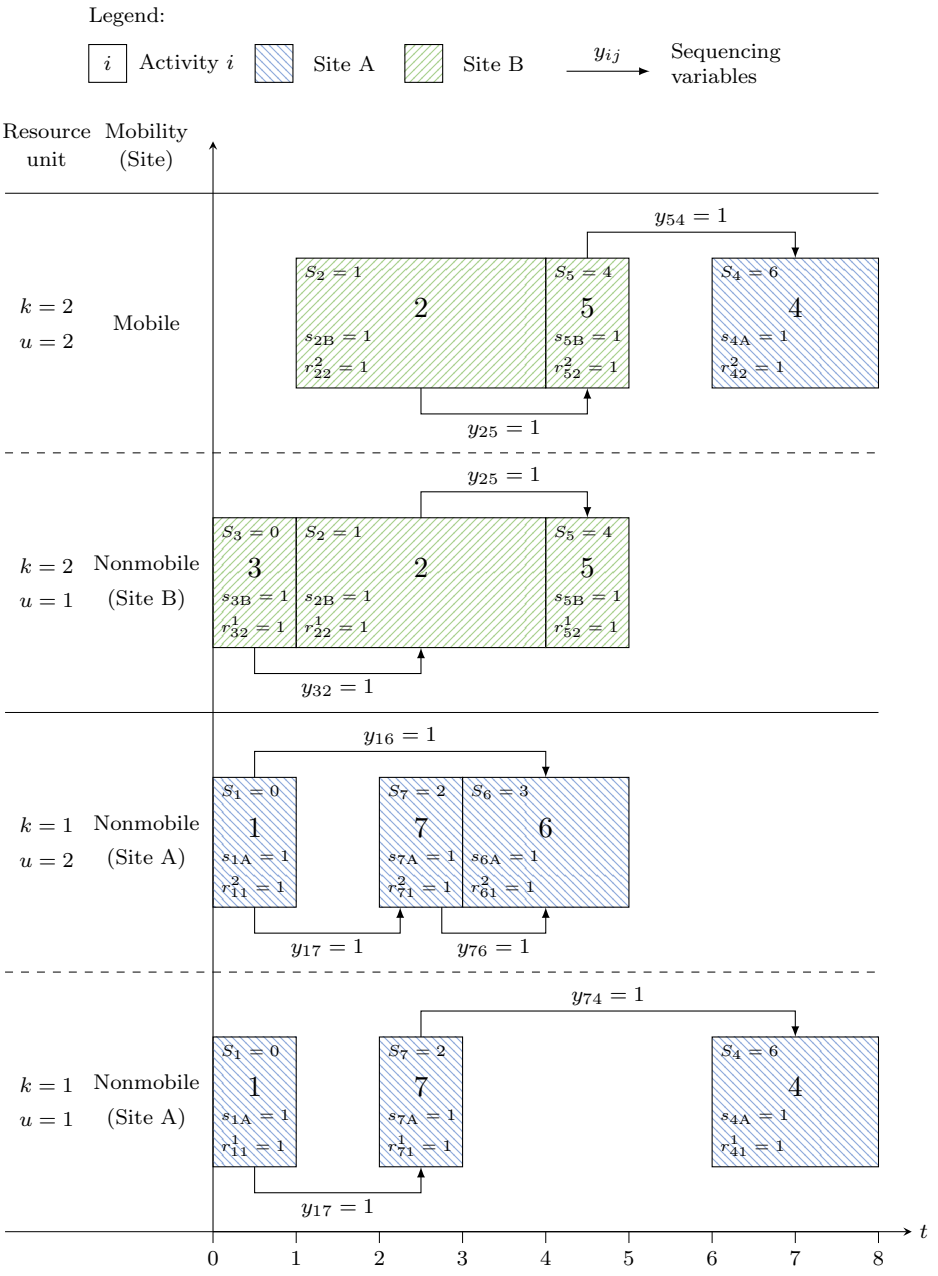

Figure 2.3: Illustration of the types of variables for the example project

activities. Moreover, a transportation time of $\delta_{ll'}$ is triggered if two precedence-related real activities i and j are performed at two different sites l and l'. This transportation time is required to move the output of the first activity between the respective sites.

$$S_i + p_i + (s_{il} + s_{jl'} - 1)\delta_{ll'} \leq S_j \quad (i,j \in V \times V :\ (i,j) \in E;\ l,l' \in L \times L) \tag{2.1}$$

Constraints (2.2) enforce that if two real activities i and j are both assigned to the same unit u of a resource type k, then either activity i must be completed before the start of activity j or vice versa, i.e., either $y_{ij} = 1$ or $y_{ji} = 1$.

$$r^u_{ik} + r^u_{jk} \leq y_{ij} + y_{ji} + 1 \quad (i,j \in V \times V;\ k \in R;\ u \in \{1,\ldots,R_k\} : \\ i < j,\ (i,j) \notin TE,\ r_{ik} > 0,\ r_{jk} > 0) \tag{2.2}$$

Constraints (2.3) link the start-time variables and the sequencing variables, where $\delta^{max} := \max_{l,l' \in L \times L}\{\delta_{ll'}\}$ corresponds to the longest transportation time between all pairs of sites. A transportation time of $\delta_{ll'}$ is triggered if two activities i and j are both assigned to at least one common resource unit and are performed at two different sites l and l'. This transportation time is required to move the commonly used resource units between the respective sites.

$$S_i + p_i + (s_{il} + s_{jl'} - 1)\delta_{ll'} \leq S_j + (\sum_{h \in V} p_h + n\delta^{max})(1 - y_{ij}) \\ (i,j \in V \times V :\ i \neq j,\ (i,j) \notin TE;\ l,l' \in L \times L) \tag{2.3}$$

Constraints (2.4) ensure that all real activities are completed before the project ends. Constraints (2.4) are required in addition to constraints (2.1) because the latter only account for the precedence relationships between all real activities, which do not include activity $n+1$.

$$S_i + p_i \leq S_{n+1} \quad (i \in V \cup \{0\}) \tag{2.4}$$

Constraints (2.5) guarantee that the number of resource units allocated to each activity is equal to the number of required resource units of the respective activity.

$$\sum_{u=1}^{R_k} r^u_{ik} = r_{ik} \quad (i \in V;\ k \in R :\ r_{ik} > 0) \tag{2.5}$$

Constraints (2.6) enforce that each activity i that is assigned to a nonmobile unit

u of any resource type k must be performed at the site at which the respective unit is permanently located.

$$r_{ik}^{u} \leq s_{i,loc_{ku}} \quad (i \in V;\ k \in R;\ u \in \{1,\ldots,R_k\}:\ M_{ku} = 0,\ r_{ik} > 0) \tag{2.6}$$

Constraints (2.7) ensure that each activity is performed at exactly one site.

$$\sum_{l \in L} s_{il} = 1 \quad (i \in V) \tag{2.7}$$

Finally, (2.8) and (2.9) are redundant constraints, where the set F^2 comprises all pairs of activities i and j that cannot be performed in parallel since together they require more resource units than are available, i.e., $r_{ik} + r_{jk} > R_k$ for some resource type k. These redundant constraints turned out to substantially speed up the solution process of the mathematical programming solver used during our computational experiment.

$$y_{ij} + y_{ji} = 1 \quad ((i,j) \in F^2) \tag{2.8}$$

$$y_{ij} + y_{ji} \leq 1 \quad (i,j \in V \times V:\ i < j,\ (i,j) \notin TE) \tag{2.9}$$

The novel model, which we refer to as the model (CT) hereafter, reads as follows:

$$(\mathrm{CT}) \begin{cases} \text{Min. } S_{n+1} \\ \text{s.t. } (2.1) - (2.9) \\ \qquad S_i \in \mathbb{R}_{\geq 0} \quad (i \in V \cup \{0, n+1\}) \\ \qquad s_{il} \in \{0,1\} \quad (i \in V;\ l \in L) \\ \qquad y_{ij} \in \{0,1\} \quad (i,j \in V \times V : i \neq j,\ (i,j) \notin TE) \\ \qquad r_{ik}^{u} \in \{0,1\} \quad (i \in V;\ k \in R;\ u \in \{1,\ldots,R_k\}:\ r_{ik} > 0) \end{cases}$$

Alternatively, the start-time variables can also be defined as nonnegative integer variables, i.e., $S_i \in \mathbb{Z}_{\geq 0}$ $(i \in V \cup \{0, n+1\})$, as long as the durations of the activities and the transportation times are all integers. In Section 2.5, we report some results of our computational experiment that indicate that this accelerates the solution process of the mathematical programming solver used.

The continuous-time model presented in this section turns out to be applicable to challenging instances comprising up to 30 activities (see Section 2.5). For instances comprising considerably more than 30 activities, a heuristic approach such as the matheuristic

presented in the following section, which is able to derive feasible solutions within a reasonable running time, may be more appropriate.

2.4 Relax-optimize-and-fix matheuristic

In this section, we present the relax-optimize-and-fix matheuristic for the multi-site RCPSP. In Section 2.4.1, we give an overview of the novel matheuristic. In Section 2.4.2, we describe the different steps of our matheuristic in detail. In Section 2.4.3, we illustrate the matheuristic with the example project from Section 2.2.

2.4.1 Overview

Matheuristics are recent approaches for combinatorial optimization problems that combine advantages of mathematical models and heuristic techniques (cf. Maniezzo et al., 2021) or metaheuristic techniques (cf. Boschetti et al., 2009). As to the four essential characteristics of accuracy, speed, simplicity and flexibility of good heuristics (cf. Cordeau et al., 2002), matheuristics especially excel regarding their simplicity and flexibility. Matheuristics are simple to implement since nowadays relatively easy-to-use modeling languages and interfaces to various programming languages are available. Moreover, matheuristics are flexible because additional constraints or alternative objective functions can easily be incorporated using mathematical modeling.

As mentioned above, our matheuristic is specifically designed for devising good feasible solutions within reasonable running time to instances of the multi-site RCPSP whose size does not permit an immediate application of the mathematical model presented in Section 2.3. To this end, we combine the mathematical model with a heuristic technique. Among the different types of matheuristics presented in Della Croce et al. (2013), our approach is related to the group of the continuous relaxation based matheuristics, which according to Della Croce et al. (2013) cleverly exploit information extracted from the LP relaxation of a mathematical model. The basic idea of our matheuristic is to iteratively schedule subsets of the activities by solving an appropriate relaxation of the mathematical model, which is obtained by relaxing the binary restrictions for subsets of the sequencing variables.

More specifically, our matheuristic runs as follows. In each iteration, a subset of the activities is scheduled while taking into account all activities of the instance. We use an appropriate variant of the model (CT) presented in Section 2.3 to schedule these activities. The matheuristic follows a relax-optimize-and-fix strategy; i.e., initially, the binary restrictions on the sequencing variables in the variant of the model (CT) are relaxed

and then iteratively restored for a subset of the activities scheduled in the next iteration. After each iteration, some activities of the subset are considered as scheduled, and hence, some variables are fixed for these activities. The other activities remain in the subset and can be rescheduled along with newly selected activities in the next iteration. An important advantage of using the model (CT) in the matheuristic is that in the model (CT), the sequence of the activities and the start times of the activities are captured using different variable types. Hence, we can fix the sequence between the scheduled activities and still keep the flexibility to move these activities along the timeline.

In our matheuristic, the trade-off between solution quality and running time can be controlled by two parameters. The first parameter indicates how many activities are scheduled in each iteration, i.e., it influences the number of variables for which the binary restrictions are restored in each iteration. In general, the solution quality improves with an increasing number of activities scheduled in each iteration, but the running time to solve the variant of the model (CT) in each iteration also increases. Therefore, this parameter trades off the solution quality against the running time in a single iteration. The second parameter indicates how many activities are considered as scheduled after each iteration and consequently how many activities are allowed to be rescheduled and how many new activities are selected. In general, the solution quality improves with a decreasing number of activities that are considered as scheduled after each iteration, but the overall running time also increases because an increasing number of iterations has to be conducted. Hence, this parameter allows us to control the number of iterations that are required to derive a feasible solution to an instance, and it trades off the solution quality against the running time over the entire process of the matheuristic.

2.4.2 Different steps of the matheuristic

In the matheuristic, set V is divided into three sets that contain the activities V^N that have not yet been scheduled, the activities V^C that are scheduled in the current iteration and the activities V^S that have been scheduled and for which several variables are fixed. Figure 2.4 shows an overview of the different steps of the matheuristic. In the following, we describe these steps in more detail.

- In step ①, the set V^N is initialized to contain all real activities, and the sets V^C and V^S are initialized as empty sets.
- In step ②, $c > 0$ activities are selected from V^N. To select these activities, a priority rule is applied which may be of the same type such as those used in, e.g., the serial schedule generation scheme. Please note that the selection of an appropriate priority

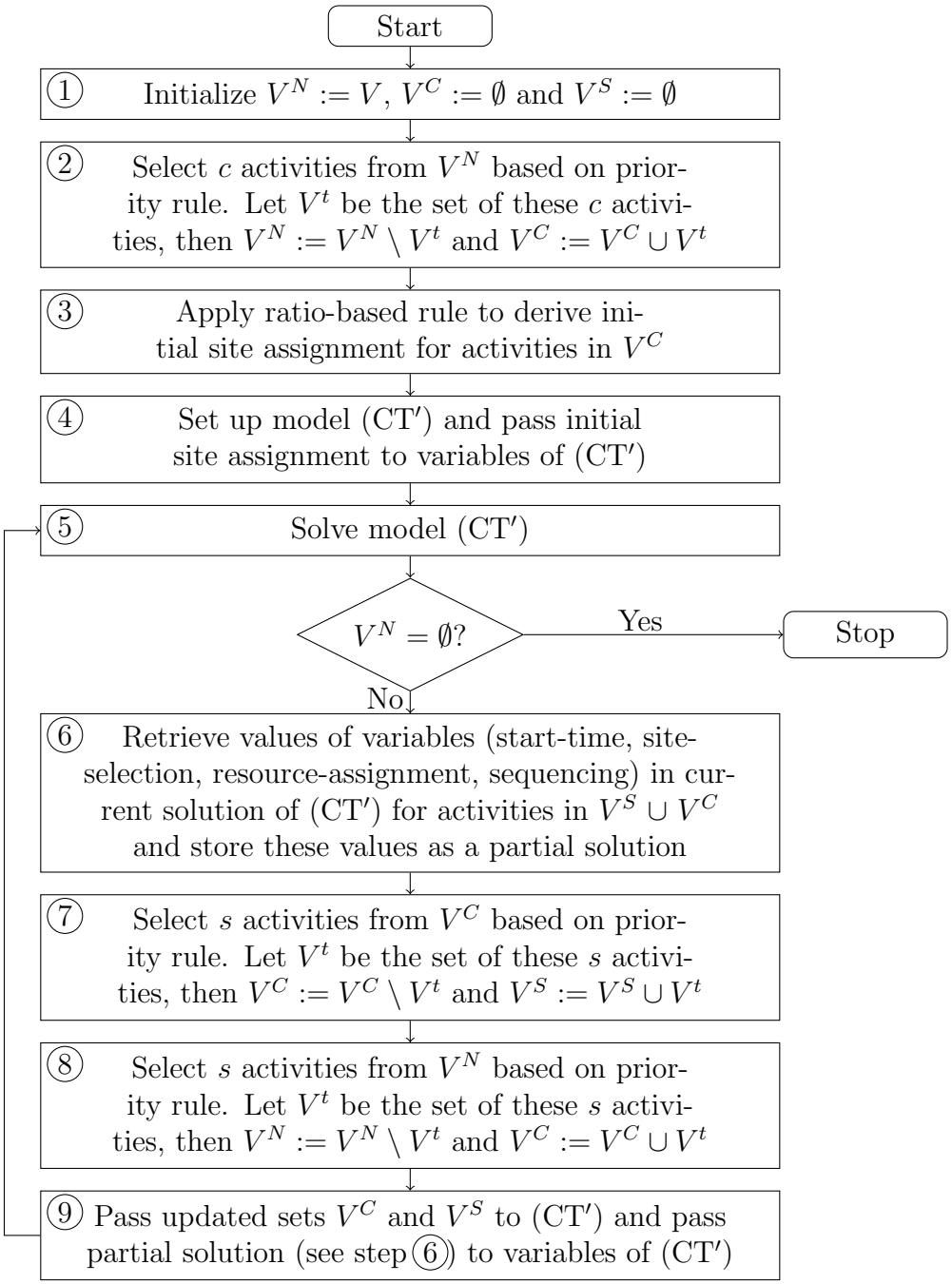

Figure 2.4: Overview of the different steps of the matheuristic

rule considerably influences the performance of the matheuristic. In Section 2.5.4, we discuss how to define a suitable value for c.

- In step ③, we derive an initial site assignment for the activities in V^C. The idea here is to select for each activity i a resource type k, of which many units are required to execute activity i and of which only a few mobile units are available; therefore, it might be promising to schedule activity i at a site where many nonmobile resource units of resource type k are available. Thus, we propose a rule that is based on a ratio between two key figures that can be computed for each activity i and each resource type k. For each activity, the resource type k^* with the largest ratio $\frac{r_{ik^*}}{1+\sum_{u=1}^{R_{k^*}} M_{k^*u}}$ is selected. Next, we determine at which site the highest number of nonmobile resource units of the selected resource type k^* are located. This site is selected as the initial site assignment of activity i. If resource type k^* has no nonmobile resource units, the site at which the highest number of nonmobile resource units are located over all resource types is selected as the initial site assignment of activity i. In the case of any ties, the respective resource or site index is used as tie-breaker. Initial testing has indicated that providing an initial site assignment by using this simple rule improves the solution quality of the matheuristic without increasing the running time. It is further important to note that the initial site assignment will be passed to the corresponding variables of a variant of the model (CT) introduced in step ④ as *initial* values, but the values of these variables may still change during the solution process.

- In step ④, a variant of the model (CT) referred to as model (CT′) is set up. The model (CT′) differs from the model (CT) in regard to several constraints and the domains of some of the variables. The binary restrictions on the sequencing variables y_{ij} are initially relaxed for all activities and restored only if both activities i and j are in $V^C \cup V^S$. In the model (CT), the redundant constraints (2.8) and (2.9) are added for all activities. However, it turns out to substantially speed up the solution process in the model (CT′) to add only the constraints in which the corresponding sequencing variables are defined as binary in each iteration instead of also adding the constraints in which the corresponding binary restrictions of the sequencing variables are relaxed. Hence, the redundant constraints (2.10) are set up for the activities (i, j) in F^2 only if these activities are in $V^C \cup V^S$:

$$y_{ij} + y_{ji} = 1 \quad ((i,j) \in F^2 :\ i,j \in V^C \cup V^S) \tag{2.10}$$

For the same reason, the second type of redundant constraints (2.11) are added only

for the activities in $V^C \cup V^S$:

$$y_{ij} + y_{ji} \leq 1 \quad (i, j \in V^C \cup V^S :\ i < j,\ (i,j) \notin TE) \tag{2.11}$$

Starting from the second iteration, some of the variables associated with the activities in V^S are fixed. Here, s^*_{il}, r^{*u}_{ik} and y^*_{ij} denote the values that the site-selection, resource-assignment and sequencing variables took in the solution to the model (CT′) in the previous iteration, respectively. Constraints (2.12) fix the site-selection variables of the activities in V^S to the values that the corresponding variables took in the last iteration.

$$s_{il} = s^*_{il} \quad (i \in V^S;\ l \in L) \tag{2.12}$$

Constraints (2.13) fix the resource-assignment variables of the activities in V^S to the values that these variables took in the last iteration.

$$r^u_{ik} = r^{*u}_{ik} \quad (i \in V^S;\ k \in R;\ u = 1, \ldots, R_k :\ r_{ik} > 0) \tag{2.13}$$

Finally, constraints (2.14) fix the sequencing variables between activities in V^S to the values that the corresponding variables took in the last iteration.

$$y_{ij} = y^*_{ij} \quad (i, j \in V^S) \tag{2.14}$$

The complete model (CT′) reads as follows:

$$(\text{CT}') \begin{cases} \text{Min. } S_{n+1} \\ \text{s.t. } (2.1) - (2.7) \\ \qquad (2.10) - (2.14) \\ \qquad S_i \in \mathbb{Z}_{\geq 0} \quad (i \in V \cup \{0, n+1\}) \\ \qquad s_{il} \in \{0,1\} \quad (i \in V;\ l \in L) \\ \qquad r^u_{ik} \in \{0,1\} \quad (i \in V;\ k \in R;\ u = 1, \ldots, R_k :\ r_{ik} > 0) \\ \qquad y_{ij} \in \{0,1\} \quad (i, j \in V^C \cup V^S :\ i \neq j,\ (i,j) \notin TE) \\ \qquad y_{ij} \in [0,1] \quad (i, j \in V :\ i \neq j,\ (i,j) \notin TE,\ \{i,j\} \not\subseteq V^C \cup V^S) \end{cases}$$

For instances that include activities with noninteger durations or noninteger transportation times, we set $S_i \in \mathbb{R}_{\geq 0}$ $(i \in V \cup \{0, n+1\})$. Moreover, we pass the initial

site assignment derived in step ③ as initial values for the corresponding variables of (CT′). Furthermore, the parameter c (see step ②) influences how many sequencing variables are defined as binary in the model (CT′). We discuss in Appendix 2.A how an appropriate value for parameter c can be derived.

- In step ⑤, the model (CT′) is solved using a generic MBLP solver with a prescribed time limit. If no feasible solution is found within the time limit, the limit is extended until a feasible solution is found. If V^N is empty, we stop; otherwise, there are still activities that need to be scheduled, and we continue with step ⑥.
- In step ⑥, a partial solution is stored based on the devised solution to the model (CT′). The partial solution comprises the values of the start-time, the site-selection and the resource-assignment variables for the activities in $V^S \cup V^C$ as well as the values of the sequencing variables between the activities in $V^S \cup V^C$ in the devised solution.
- In step ⑦, s activities with $0 < s \leq c$ are selected from V^C based on the priority rule from step ②. These activities are then removed from V^C and included in V^S. For these activities, some variables are fixed in the next iteration (see constraints (2.12)–(2.14)).
- In step ⑧, s activities are selected from V^N based on the priority rule from step ②. These activities are removed from V^N and included in V^C to be scheduled in the next iteration. If there are fewer than s activities in V^N, all activities in V^N are selected.
- In step ⑨, because V^C was updated in steps ⑦ and ⑧, some binary restrictions that were relaxed are now restored (see the domains of the sequencing variables in (CT′)). Moreover, as V^S was updated in step ⑦, some variables that were not fixed are now fixed to their corresponding values (see constraints (2.12)–(2.14)). Finally, some redundant constraints are added (see constraints (2.10)–(2.11)). Next, the partial solution from step ⑥ is provided as initial values for the variables of the model (CT′) in the same way as the initial site assignment in step ④.

The matheuristic stops when—after step ⑤ is completed—V^N is empty and, thus, a feasible solution has been derived.

Table 2.2: Latest start times (LST) and latest finish times (LFT) of the activities in the example project

Activity	1	2	3	4	5	6	7
LST	2	0	2	3	4	3	4
LFT	3	3	3	5	5	5	5

2.4.3 Illustration of the matheuristic

We illustrate the matheuristic with the example project from Section 2.2 with $c = 4$ and $s = 3$. First, the sets $V^S = \emptyset$, $V^C = \emptyset$, and $V^N = \{1, 2, 3, 4, 5, 6, 7\}$ are initialized.

Second, $c = 4$ activities are selected from V^N. Here, we apply the latest start time priority rule with the latest finish time priority rule as a tie breaker (LST + LFT); i.e., the activity with the smallest latest start time (and, in case of a tie, with the smallest latest finish time) is selected first. The latest start time and the latest finish time of each real activity are determined by (forward and) backward pass calculation (cf. Demeulemeester and Herroelen, 2002). The latest start time and the latest finish time are standard priority rules that have been shown to perform well in the recent scheduling literature (cf., e.g., Almeida et al., 2016) and perform well in our matheuristic. Table 2.2 states the latest start time and the latest finish time of each real activity, which results in $V^C = \{2, 1, 3, 4\}$ and $V^N = \{5, 6, 7\}$.

Third, the initial site assignment is derived by the ratio-based rule. The rule derives site A as the initial site for activities 1 and 4 and site B as the initial site for activities 2 and 3. For activity 4, for example, the resource type $k^* = 1$ is selected, since it has the largest ratio $\frac{r_{41}}{1+\sum_{u=1}^{R_1} M_{1u}} = \frac{1}{1+0} = 1$. Resource type $k^* = 1$ has two nonmobile resource units located at site A but none at site B. Thus, site A is selected as the initial site for activity 4.

Fourth, the model (CT′) is set up, and the initial site assignment is passed to the variables of the model (CT′).

Fifth, the (CT′) is solved with the Gurobi solver and a time limit of 60 seconds. Figure 2.5 (top) shows the optimal solution obtained in the first iteration. The activities in V^C are highlighted in bold and the activities in V^N are highlighted as slightly transparent. Two pairs of activities $\{2, 5\}$ and $\{6, 7\}$ are executed in parallel, which is feasible because the binary restrictions on the sequencing variables between these pairs of activities were relaxed in the model (CT′) in the first iteration.

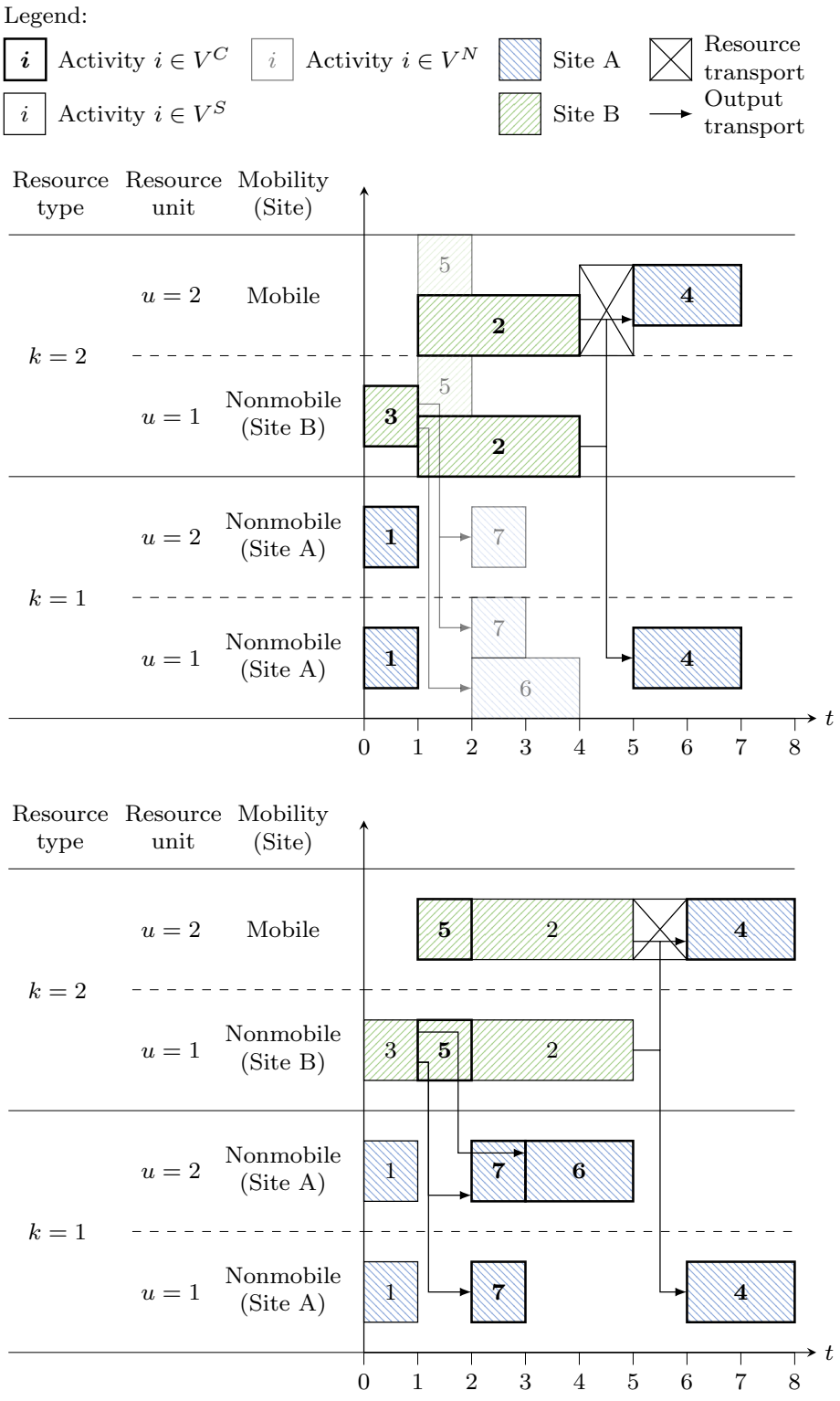

Figure 2.5: Schedule for the example project derived by the matheuristic after Iteration 1 (top) and after Iteration 2 (bottom)

Sixth, the partial solution based on the current solution to the model (CT′) is stored. For example, for the site-selection variable s_{4A}, the value of 1 is stored.

Seventh, the $s = 3$ activities with the highest priority among the c activities in V^C are removed from V^C and included in V^S. In the first iteration, these are the activities $\{2, 1, 3\}$.

Eighth, $s = 3$ activities with the highest priority among the activities in V^N are removed from V^N and included in V^C. These are the activities $\{6, 5, 7\}$.

In Iteration 2, the activities $V^C = \{4, 6, 5, 7\}$ must be scheduled. Figure 2.5 (bottom) shows the optimal solution obtained in the second iteration. The activities V^S are not highlighted. No activities are executed in parallel because all activities are either in V^C or V^S and, thus, all sequencing variables are defined as binary. Activity 2 is scheduled one time unit later than in Iteration 1 even though it is in V^S. This is possible because only the sequencing variables between all activities in V^S are fixed, but not their start time variables. After Iteration 2, the matheuristic stops, and a feasible solution has been found that also turns out to be an optimal solution for our example project.

2.5 Computational results

In this section, we provide the results of our computational experiments in which we compare the model presented in Section 2.3 and the matheuristic presented in Section 2.4 with the state-of-the-art approaches from the literature presented by Laurent et al. (2017). In Section 2.5.1, we describe the experimental design, and in Section 2.5.2, we describe the metrics used to evaluate the performance of the examined approaches. In Section 2.5.3, we compare the performance of three exact approaches based on the presented continuous-time model and the discrete-time model proposed by Laurent et al. (2017) using a standard mathematical programming solver. In Section 2.5.4, we analyze the trade-off between solution quality and running time that can be controlled by setting the parameter values of the presented matheuristic, and we compare its performance to the performance of the four metaheuristics of Laurent et al. (2017).

2.5.1 Experimental design

We implemented all models and our matheuristic in Python 3.7, and as mathematical programming solver we used the Gurobi Optimizer 9.1 with the default solver settings. All computations were performed on an HP workstation with one Intel Xeon CPU with 2.20 GHz clock speed and 128 GB RAM. For the three exact approaches, we set a time limit of 300 seconds. For the matheuristic, we used the same parameter setting as described

in Section 2.4.3 except for the values of the parameters c and s (see Section 2.5.4 and Appendix 2.A). The initial values passed to the model (CT′) in the matheuristic are provided to the Gurobi Optimizer by the start attribute of the variables (cf. `https://www.gurobi.com/documentation/9.1/refman/start.html`). To ensure a fair comparison of the performance, we also ran the implementation of the four metaheuristics of Laurent et al. (2017) on the same workstation; this implementation was kindly provided by the authors. Moreover, Laurent et al. (2017) adjusted the test instances of the sets j30 and j60 from the PSPLIB (cf. Kolisch and Sprecher, 1996) to the multi-site context and named these newly generated sets MSj30 and MSj60, respectively. We used the same sets of instances for our experiments.

2.5.2 Description of performance metrics

To evaluate the performance of the tested exact and heuristic approaches, we use the following metrics:

- #Feas: Number of instances for which a feasible schedule is found within the prescribed time limit.
- #Opt: Number of instances for which a feasible schedule is found and proved to be optimal within the prescribed time limit.
- Gap^{CP} (%): Average relative deviation between the objective function value of the best solution returned by the respective approach (OFV) and the critical-path-based lower bound (CP), calculated as $(OFV - CP)/CP$.
- CPU (s): Average running time to derive the best solution returned by the respective approach.
- $\#\text{MH}^{+}$: Number of instances for which the matheuristic finds a solution with a lower OFV than the respective metaheuristic.
- $\#\text{MH}^{-}$: Number of instances for which the matheuristic finds a solution with a higher OFV than the respective metaheuristic.

In all tables, bold values indicate the best results among all tested approaches.

2.5.3 Computational results: Exact approaches

In Table 2.3, we report the results of the exact approaches based on the model (CT), the model (CT) with integer start-time variables, subsequently referred to as model (CT^{int}),

Table 2.3: Results for all MSj30 instances with the models (CT), (CT^{int}) and (DT)

Activities	Sites	Model	#Feas	#Opt	Gap^{CP} (%)	CPU (s)
30	2	(CT)	**480**	327	25.86	112.00
		(CT^{int})	**480**	**331**	**25.17**	**110.83**
		(DT)	455	272	34.93	159.44
30	3	(CT)	**480**	284	33.92	**138.40**
		(CT^{int})	**480**	**289**	**33.28**	139.25
		(DT)	444	224	51.80	190.93

Table 2.4: Results for the MSj30 instances for which with each of the models (CT), (CT^{int}) and (DT) at least a feasible solution was obtained

Activities	Sites	Model	#Feas	#Opt	Gap^{CP} (%)	CPU (s)
30	2	(CT)	455	323	22.04	102.92
		(CT^{int})	455	**325**	**21.35**	**102.80**
		(DT)	455	272	34.93	151.71
30	3	(CT)	444	279	28.51	**127.50**
		(CT^{int})	444	**284**	**27.91**	128.25
		(DT)	444	224	51.80	182.08

and the discrete-time model of Laurent et al. (2017), subsequently referred to as model (DT), for all MSj30 instances. In addition, in Table 2.4, we present the corresponding results for the MSj30 instances for which each approach obtains at least a feasible solution.

With the model (CT), a feasible solution for all MSj30 instances is found. In contrast, with the model (DT), no feasible solutions are found for numerous instances. In terms of all other performance metrics, the model (CT) outperforms the model (DT); i.e., with the model (CT), a higher number of optimal solutions are derived, a lower average gap to the critical-path-based lower bound is achieved, and a lower average running time is required. Moreover, our results indicate that the problem becomes more challenging as the number of sites increases. We find that defining the start-time variables of the model (CT) as integer variables further improves its performance.

2.5.4 Computational results: Heuristic approaches

In this section, we analyze an important feature of our matheuristic, namely the possibility to control the trade-off between solution quality and computation time. Moreover, we compare the performance of our matheuristic to the performance of the state-of-the-art heuristic approaches from the literature.

First, we examine the trade-off between solution quality and running time that can be controlled by the two parameters c and s of our matheuristic. Recall that parameter c indicates how many activities are scheduled in each iteration, and parameter s indicates how many activities are considered as scheduled after each iteration. To examine this trade-off, we ran our matheuristic for various values of parameter s, while fixing parameter c to a value of $c = 12$. For the sake of readability, we explain how we derived this value for parameter c in Appendix 2.A.

In Table 2.5, we report the results of the matheuristic with these parameter settings for all MSj30 and MSj60 instances. The results indicate that decreasing parameter s improves the solution quality considerably at the expense of additional running time. Moreover, the results show that our matheuristic finds a feasible solution for all MSj30 and MSj60 instances within a reasonable running time. Additionally, compared to the exact approach based on the model (CT) with the imposed time limit of 300 seconds, the matheuristic on average derives solutions with a lower objective function value, while its average running time is considerably lower. We further evaluated the percentage of the running time that the matheuristic spends in the mathematical programming component (i.e., the total time used to set up and solve model (CT′)) versus the percentage of the running time it spends in the heuristic component (e.g., to apply the priority rule to select the first c or the next s activities). Overall, more than 99% of the total running time is spent in the mathematical programming component.

Second, we compare the performance of our matheuristic to the performance of the following four metaheuristics of Laurent et al. (2017): local search (LS), simulated annealing (SA), iterated local search with a better walk acceptance criterion (ILS BW) and iterated local search with a simulated annealing acceptance criterion (ILS SA). For a fair comparison, we use for each combination of the number of activities (30 and 60) and the number of sites (2 and 3) in the instances the parameter setting of the matheuristic for which the average running time is the closest to the average running times of the metaheuristics.

In Table 2.6, we report the results of the matheuristic with these parameter settings and the four metaheuristics for all MSj30 and MSj60 instances. For the MSj30 instances, the average running time of the matheuristic is slightly higher than the average running

Table 2.5: Results for all MSj30 and MSj60 instances for the matheuristic (MH) for various values of s

Activities	Sites	Parameters of MH		#Feas	Gap^{CP} (%)	CPU (s)
30	2	$c = 12$	$s = 2$	480	**25.02**	61.86
			$s = 4$	480	25.48	36.75
			$s = 6$	480	26.01	28.81
			$s = 8$	480	27.13	24.39
			$s = 10$	480	27.98	21.14
			$s = 12$	480	29.48	**18.81**
30	3	$c = 12$	$s = 2$	480	**31.59**	102.57
			$s = 4$	480	32.35	61.32
			$s = 6$	480	32.86	47.97
			$s = 8$	480	34.21	38.83
			$s = 10$	480	34.90	34.07
			$s = 12$	480	37.20	**29.76**
60	2	$c = 12$	$s = 2$	480	**23.17**	277.73
			$s = 4$	480	23.91	164.50
			$s = 6$	480	24.99	123.18
			$s = 8$	480	25.59	100.40
			$s = 10$	480	26.62	88.50
			$s = 12$	480	28.05	**83.45**
60	3	$c = 12$	$s = 2$	480	**31.09**	375.90
			$s = 4$	480	32.52	221.63
			$s = 6$	480	33.45	166.20
			$s = 8$	480	35.42	137.08
			$s = 10$	480	38.14	122.94
			$s = 12$	480	39.52	**117.38**

Table 2.6: Comparison of the matheuristic to the state-of-the-art heuristics for all MSj30 and MSj60 instances

Activities	Sites	Approach	#Feas	Gap^{CP} (%)	$\#\text{MH}^{+}$	$\#\text{MH}^{-}$	CPU (s)
30	2	MH ($c = 12$, $s = 2$)	480	**25.02**	0	0	61.86
		LS	480	29.72	258	61	**55.50**
		SA	480	26.50	178	93	55.51
		ILS LS	480	25.86	153	103	70.35
		ILS SA	480	26.04	152	110	70.80
30	3	MH ($c = 12$, $s = 4$)	480	**32.35**	0	0	61.32
		LS	480	37.65	276	89	**60.17**
		SA	480	34.11	219	119	60.44
		ILS LS	480	33.42	180	142	76.53
		ILS SA	480	33.37	192	152	76.42
60	2	MH ($c = 12$, $s = 6$)	480	**24.99**	0	0	**123.18**
		LS	480	27.95	231	133	128.97
		SA	480	26.19	211	168	130.26
		ILS LS	480	26.41	198	163	168.71
		ILS SA	480	26.57	197	161	168.68
60	3	MH ($c = 12$, $s = 8$)	480	**35.42**	0	0	**137.08**
		LS	480	38.29	289	123	142.76
		SA	480	35.51	235	172	143.66
		ILS LS	480	36.03	242	172	185.79
		ILS SA	480	36.43	249	149	185.98

time of some of the metaheuristics, while the matheuristic has on average a better solution quality than do the four metaheuristics. For the MSj60 instances, the matheuristic has a shorter average running time than all four metaheuristics and derives solutions with a lower average gap to the critical-path-based lower bound. Moreover, we can see from Table 2.6 that for a large number of instances, the matheuristic derives a solution with a lower objective function value than the metaheuristics.

2.6 Conclusion

In this paper, we studied a variant of the resource-constrained project scheduling problem in which activities can be scheduled at different sites and some of the resource units can be transported between sites. We proposed a continuous-time model and a matheuristic for this planning problem. The continuous-time model is based on continuous variables that

represent the start times of the activities and binary variables that represent the selection of a site for the activities, the assignment of the activities to the resource units and the sequence between the activities if the activities are assigned to at least one common resource unit. The matheuristic is based on a variant of the continuous-time model in which the binary restrictions on the sequencing variables are initially relaxed for all activities and then iteratively restored for a subset of the activities that is scheduled in the current iteration. In each iteration of the matheuristic, all activities are taken into account when the respective subset of activities is scheduled. The model and the matheuristic outperform state-of-the-art exact and heuristic approaches from the literature in terms of various performance measures on a set of standard instances.

In future research, the developed approaches could be used to further analyze the benefit of pooling resources among different sites. Furthermore, for instances with substantially more than 60 activities, the time used to solve the variant of the continuous-time model in each iteration of the matheuristic may strongly increase. Thus, we suggest dividing these instances into several smaller instances and successively deriving a schedule for these smaller instances. Finally, similar matheuristics based on the relax-optimize-and-fix strategy could be designed for related resource-constrained project scheduling problems. For the multi-mode RCPSP, for example, the sequencing between activities could be similarly disregarded in a first step and then iteratively determined.

Appendix

2.A Appendix A

We derived the value of parameter c used for the computational analysis in Section 2.5.4 by running our matheuristic for various values of parameter c, while fixing parameter s to a value of $s = 2$. We fixed the value of parameter s to a small value, since small values generally lead to a better solution quality than large values (at the expense of a longer running time). To derive a value for parameter c, we selected the largest instances comprising 60 activities and 3 sites.

In Table 2.A.1, we report the results of this examination. When parameter c is increased, the solution quality first improves and then deteriorates again while the average running time increases consistently. From the tested values of parameter c, we selected $c = 12$ because this value resulted in the best solution quality.

In general, the solution quality improves for increasing values of parameter c at the expense of additional running time. The deterioration of the solution quality that we observe

Table 2.A.1: Results for the MSj60 instances comprising 3 sites for the matheuristic for various values of c

Activities	Sites	c	#Feas	Gap^{CP} (%)	CPU (s)
60	3	8	480	33.90	230.04
		12	480	31.09	375.90
		16	480	31.21	470.80
		20	480	33.72	532.09

for large values of parameter c, however, is due to the running time limit of 60 seconds imposed in each iteration of the matheuristic. We imposed this running time limit to prevent excessively long total running times. However, for large values of parameter c, this running time limit may force the solution process of model (CT$'$) to be terminated prematurely which may lead to a poor solution quality.

Bibliography

Almeida, B.F., Correia, I., Saldanha-da-Gama, F., 2016. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. Expert Systems With Applications 57, 91–103.

Boschetti, M.A., Maniezzo, V., Roffilli, M., Bolufé Röhler, A., 2009. Matheuristics: Optimization, simulation and control, in: Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (Eds.), International Workshop on Hybrid Metaheuristics. Springer, pp. 171–177.

Cordeau, J.F., Gendreau, M., Laporte, G., Potvin, J.Y., Semet, F., 2002. A guide to vehicle routing heuristics. Journal of the Operational Research Society 53(5), 512–522.

Della Croce, F., Grosso, A.C., Salassa, F., 2013. Matheuristics: embedding MILP solvers into heuristic algorithms for combinatorial optimization problems, in: Siarry, P. (Ed.), Heuristics: Theory and Applications. Nova Science Publishers, New York, pp. 53–67.

Demeulemeester, E.L., Herroelen, W., 2002. Project Scheduling: a Research Handbook. Kluwer Academic Publishers, Boston.

Gnägi, M., Trautmann, N., 2019. A continuous-time mixed-binary linear programming formulation for the multi-site resource-constrained project scheduling problem, in: Wang, M., Li, J., Tsung, F., Yeung, A. (Eds.), Proceedings of the 2019 IEEE International Conference on Industrial Engineering and Engineering Management, Macau. pp. 611–614.

Gnägi, M., Rihm, T., Zimmermann, A., Trautmann, N., 2019. Two continuous-time assignment-based models for the multi-mode resource-constrained project scheduling problem. Computers & Industrial Engineering 129, 346–353.

Kadri, R., Boctor, F., 2014. Multi-mode resource-constrained project scheduling with sequence dependent transfer times, in: Fliedner, T., Kolisch, R., Naber, A. (Eds.),

14th International Conference on Project Management and Scheduling, Munich. pp. 116–119.

Kadri, R., Boctor, F., 2018. An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case. European Journal of Operational Research 265(2), 454–462.

Kolisch, R., Sprecher, A., 1996. PSPLIB—a project scheduling problem library. European Journal of Operational Research 96(1), 205–216.

Krüger, D., Scholl, A., 2009. A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. European Journal of Operational Research 197(2), 492–508.

Krüger, D., Scholl, A., 2010. Managing and modelling general resource transfers in (multi-)project scheduling. OR Spectrum 32(2), 369–394.

Laurent, A., Deroussi, L., Grangeon, N., Norre, S., 2017. A new extension of the RCPSP in a multi-site context: Mathematical model and metaheuristics. Computers & Industrial Engineering 112, 634–644.

Liu, Y., Zhou, J., Lim, A., Hu, Q., 2021. Lower bounds and heuristics for the unit-capacity resource constrained project scheduling problem with transfer times. Computers & Industrial Engineering 161, 107605.

Maniezzo, V., Boschetti, M.A., Stützle, T., 2021. Preface, in: Maniezzo, V., Boschetti, M.A., Stützle, T. (Eds.), Matheuristics. Springer, Cham, pp. 143–158.

Mika, M., Waligora, G., Weglarz, J., 2008. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research 187(3), 1238–1250.

Poppenborg, J., Knust, S., 2016. A flow-based tabu search algorithm for the RCPSP with transfer times. OR Spectrum 38(2), 305–334.

Rihm, T., Trautmann, N., 2017. An assignment-based continuous-time MILP model for the resource-constrained project scheduling problem, in: De Meyer, A., Chai, K.H., Jiao, R., Chen, N., Xie, M. (Eds.), Proceedings of the 2017 IEEE International Conference on Industrial Engineering and Engineering Management, Singapore. pp. 35–39.

Sabzehparvar, M., Seyed-Hosseini, S.M., 2008. A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. The Journal of Supercomputing 44(3), 257–273.

Paper III

A matheuristic for locating obnoxious facilities[3]

Tamara Bigler

Department of Business Administration
University of Bern

Contents

[3]Paper under review, submitted to Computers & Operations Research

Abstract

Facilities such as waste plants or wind turbines are often referred to as obnoxious facilities because they negatively affect their nearby environment, for example, through noise or pollution. In the obnoxious p-median problem, a set of clients and a set of potential locations for obnoxious facilities are given. From the set of potential locations, p facilities must be opened. The goal is to place the facilities far away from clients to avoid high negative effects on them. Existing approaches for this planning problem are either not scalable to large instances or not flexible in considering practical constraints that often occur in real-world settings. To address these limitations, we propose a matheuristic for this planning problem that scales to instances involving thousands of clients and potential locations and is flexible to incorporate practical constraints. Our computational results show that the matheuristic outperforms the leading metaheuristics from the literature on large instances and is competitive with the leading metaheuristics on small and medium instances.

3.1 Introduction

Facility location problems typically consist of placing facilities in a given space (cf. ReVelle and Eiselt, 2005). Depending on the facilities' characteristics, different objectives arise. Some facilities are desirable to the nearby population because they generate added value and should thus be located close to the population, i.e., the clients. Other facilities are obnoxious because they have adverse effects on their close surroundings and should therefore be placed away from the population. Examples of obnoxious facilities are waste plants, wind turbines, and quarantine sites. The undesirable effects include odor nuisance in the case of a waste plant, noise in the case of a wind turbine, and risk of infection in the case of a quarantine site. The location decisions of these facilities have far-reaching consequences for the population because an obnoxious facility may impact the nearby population's health or life quality and substantially influences real estate prices.

An often-studied planning problem is the obnoxious p-median problem (OPMP), which we consider in this paper. In the OPMP, a set of clients and a set of potential locations for obnoxious facilities are given. From the set of potential locations, p facilities must be selected to be open. For each client, the distance to the nearest open facility must be determined. The sum of these distances must be maximized. Although clients do not

necessarily receive services from the obnoxious facilities, the term *client* is common in the literature. Therefore, we also refer to them as clients in this paper. In this planning problem, a client is assumed to be only affected by the undesirable effects of the nearest open facility. For example, when a facility emits an unpleasant odor, the main contributor to the strength of the odor is assumed to be the client's nearest open facility.

Several exact approaches for the OPMP have been recently introduced. Of the exact models, the model of Lin and Chiang (2021) has been shown to deliver the best performance. It derives good feasible solutions to instances involving up to 300 clients and 300 potential locations for facilities in reasonable running time. For larger instances, exact approaches are unsuitable due to their limited scalability. To address the issue of scalability, various metaheuristics have been introduced for the OPMP. The two leading metaheuristics have been proposed by Gokalp (2020) and Chang et al. (2021) and have been applied to instances comprising up to 450 clients and 450 potential locations for facilities. Since real-world instances can involve a substantially higher number of clients and potential locations (cf. Kalczynski et al., 2022), it is important to evaluate the scalability of these metaheuristics on even larger instances. A drawback of existing metaheuristics is their lack of flexibility in incorporating practical constraints. In practical settings, it is often the case that additional constraints must be enforced. For example, authorities may impose an upper bound on the number of open facilities in specific regions. While exact approaches can easily be adjusted to incorporate practical constraints, the existing metaheuristics are difficult to adjust. Hence, none of the existing approaches are both scalable and flexible, which is why we propose here a matheuristic that combines the flexibility of exact approaches with the scalability of metaheuristics.

The proposed matheuristic consists of an initialization procedure and an improvement procedure. The initialization procedure constructs a set of diverse initial solutions. The improvement procedure improves the initial solutions by iteratively removing and adding a given number of facilities. The removal and addition of facilities is accomplished with two adapted versions of the model of Lin and Chiang (2021). The main methodological contributions are fourfold. First, we design the initialization procedure such that the generated solutions are guaranteed to be very different from each other. This diversification ensures that the solution space is explored effectively. Second, we introduce enhancements for the model formulations used in the improvement procedure. These enhancements substantially reduce the running time for setup and solution of the models. Third, the improvement procedure generates new subproblems based on information from previous subproblems, which increases the likelihood of finding large improvements early in the search process. Fourth, for instances involving a vast number of clients, we

additionally present a scaling technique that is based on the clustering of clients. All these contributions are useful for the development of matheuristics for related planning problems.

We compare the performance of our matheuristic to the performance of the two leading metaheuristics from Gokalp (2020) and Lin and Chiang (2021) on well-known benchmark instances from the literature and new instances that we introduce in this paper. Our proposed matheuristic outperforms the leading metaheuristics on large instances comprising thousands of clients and potential locations for facilities. Moreover, it is competitive with both metaheuristics for small and medium instances. Finally, we show that practical constraints can be easily incorporated in our matheuristic.

The rest of this paper is structured as follows. In Section 3.2, we describe the planning problem in more detail. In Section 3.3, we provide an overview of the most closely related literature. In Section 3.4, we present the proposed matheuristic. In Section 3.5, we report the computational results for the instances of the OPMP. In Section 3.6, we introduce an extension of the OPMP and provide results of the matheuristic for selected instances. Finally, in Section 3.7, we give some concluding remarks and an outlook on promising directions for future research.

3.2 Obnoxious p-median problem

In this section, we describe the planning problem in more detail (cf. Section 3.2.1) and illustrate it by means of a small example (cf. Section 3.2.2).

3.2.1 Problem description

The planning problem can be stated as follows (cf., e.g., Lin and Chiang, 2021). Given are a set of n clients denoted as $I = \{1, \ldots, n\}$ and a set of m potential locations for facilities denoted as $J = \{1, \ldots, m\}$ and subsequently referred to as potential facilities. Each client $i \in I$ and each potential facility $j \in J$ corresponds to a point in a plane $v_i \in \mathbb{R}^2$ and $w_j \in \mathbb{R}^2$, respectively. The parameter d_{ij} denotes the distance (e.g., Euclidean) between each client i and each potential facility j that can be computed based on the respective coordinates. Please note that the matheuristic introduced in Section 3.4 is also applicable to instances where only the distances d_{ij} but not the coordinates are available. For each client i, the distance D_i to the nearest open facility must be determined. The goal is to open $p < m$ facilities such that the sum of the distances D_i is maximized. Tamir (1991) show that this planning problem is NP-hard.

Lin and Chiang (2021) introduce a mixed-integer program for the OPMP, subsequently referred to as model (OM). The variables are described in Table 3.2.1. For each client i, a facility sequence F_i is derived in which the potential facilities are sorted in ascending order of the distances d_{ij}. The index of a potential facility at position k of the sequence is denoted by f_{ik} and $F_i = \{f_{ik} : k = 1, \ldots, m\}$.

$$
(\mathrm{OM}) \begin{cases}
\text{Max.} \quad \displaystyle\sum_{i \in I} D_i & & (3.1) \\
\text{s.t.} \quad \displaystyle\sum_{j \in J} y_j = p & & (3.2) \\
D_i \leq d_{if_{im}} - \displaystyle\sum_{k<m} (d_{if_{i,k+1}} - d_{if_{ik}}) v_{if_{ik}} & (i \in I) & (3.3) \\
v_{if_{ik}} \leq v_{if_{i,k+1}} & (i \in I, k < m) & (3.4) \\
\displaystyle\sum_{i \in I} v_{ij} \geq n y_j & (j \in J) & (3.5) \\
y_j \in \{0, 1\} & (j \in J) & (3.6) \\
v_{ij} \in \{0, 1\} & (i \in I, j \in J) & (3.7) \\
D_i \geq 0 & (i \in I) & (3.8)
\end{cases}
$$

In the objective function (3.1), the sum of the distances from each client i to the client's nearest open facility is maximized. Constraint (3.2) ensures that exactly p facilities are opened. Constraints (3.3) pose an upper bound on the distance D_i from each client i to the client's nearest open facility in the form of a telescopic sum. Constraints (3.4) force the variable v_{ij} to take the value one if facility j is further away from client i than at least one open facility. Constraints (3.5) ensure that the variables v_{ij} take the value of one for each client i if facility j is opened. Finally, constraints (3.6) and (3.7) define the domains of the binary variables, and constraints (3.8) define the domains of the continuous variables.

3.2.2 Illustrative example

We present a small example to illustrate the planning problem described in Section 3.2.1. Given is a set of $n = 45$ clients and a set of $m = 45$ potential facilities. In total, $p = 8$ facilities must be opened. Figure 3.2.1 shows the coordinates of all clients and potential facilities and an optimal solution to the example. The distances d_{ij} between each client i and each potential facility j are computed using the Euclidean distance. The distance to the nearest open facility for each client is highlighted with a gray line. The objective

Table 3.2.1: Variable descriptions

Variable	Description
D_i	Distance of client i to nearest open facility
y_j	$\begin{cases} = 1, \text{ if facility } j \text{ is open} \\ = 0, \text{ otherwise} \end{cases}$
v_{ij}	$\begin{cases} = 1, \text{ if facility } j \text{ is open or further away from client } i \text{ than at least one open facility} \\ = 0, \text{ otherwise} \end{cases}$

function value of this optimal solution is 455.7.

3.3 Literature

In the literature, various obnoxious facility location problems have been studied (cf. Church and Drezner, 2022 for an overview). Here, we focus on the literature on the OPMP. In Section 3.3.1, we present the exact solution approaches from the literature, and in Section 3.3.2, we discuss the metaheuristic approaches that have been introduced for the OPMP. To the best of our knowledge, these are the only types of solution approaches that exist for the OPMP. Readers interested in extensions of the OPMP are referred to the following articles: Colmenar et al. (2018), Sánchez-Oro et al. (2022), Kalczynski et al. (2020), Kalczynski and Drezner (2021), and Kalczynski and Drezner (2022).

3.3.1 Exact approaches

Labbé et al. (2001) introduce the OPMP and provide a proof of its NP-hardness. In Belotti et al. (2007), the same authors introduce a binary linear program (BLP) for the OPMP and present a branch-and-cut approach based on several valid inequalities. They apply their branch-and-cut approach to instances including up to 300 clients and 300 potential facilities and show that their approach outperforms the BLP. Many instances involving 100 clients and 100 potential facilities or more cannot be solved by the branch-and-cut approach within 4 hours of running time.

Chiang and Lin (2017) introduce a compact, mixed-integer linear program for the OPMP. They prove its equivalence to the model of Belotti et al. (2007) and apply their model to instances comprising up to 200 clients and 200 potential facilities. The compact

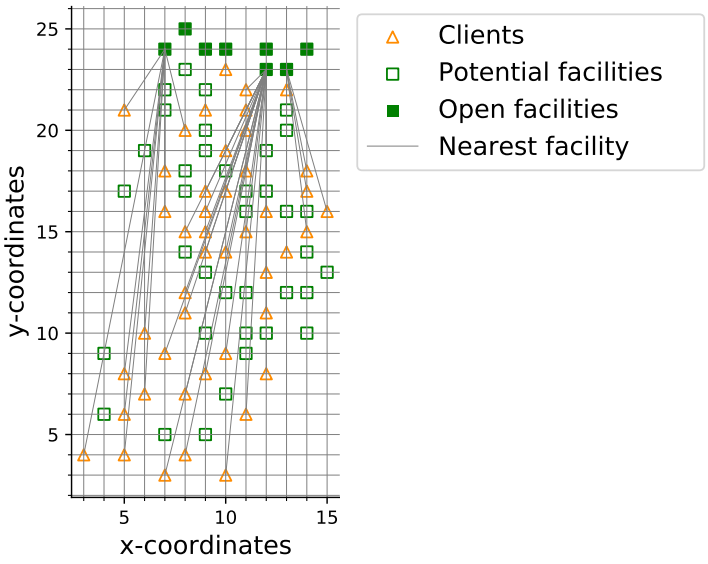

Figure 3.2.1: An optimal solution for the illustrative example

model uses considerably less running time to solve most instances. However, it still could not solve the largest considered instance within 12 hours of running time.

Lin and Chiang (2021) introduce two alternative mixed-integer linear programs for the OPMP. The first model is based on the unified model of Lei and Church (2015) for the unweighted partial-sum dispersion problem. The second model utilizes the sorting of distances between clients and potential facilities. Lin and Chiang (2021) apply the models to instances comprising up to 200 clients and 200 potential facilities. They show that the model based on sorting distances outperforms all previously introduced models for the OPMP. Hence, this model represents the state-of-the-art among the exact approaches.

3.3.2 Metaheuristic approaches

Colmenar et al. (2016) develop a metaheuristic for the OPMP based on a greedy randomized adaptive search procedure (GRASP). They introduce two constructive procedures in which facilities are iteratively added to a partial solution. They further present two improvement strategies that apply a facility exchange move. Their GRASP metaheuristic outperforms the branch-and-cut approach of Belotti et al. (2007) (applied with a time limit) in terms of solution quality and running time on instances comprising up to 450 clients and 450 potential facilities. The instances introduced by Colmenar et al. (2016) are also used in the subsequently discussed papers to evaluate the performance of the metaheuristics.

Lin and Guan (2018) propose a hybrid binary particle swarm optimization (HBPSO). The metaheuristic approach combines particle swarm optimization with an iterated greedy local search. They introduce a new position updating rule designed to inherit the good structure of previous good solutions. On average, their HBPSO metaheuristic derives solutions with a slightly better objective function value than the GRASP heuristic of Colmenar et al. (2016) in shorter running times.

Herrán et al. (2020) introduce a parallel variable neighborhood search (PVNS) metaheuristic for solving the OPMP. One of the main features of their PVNS metaheuristic is to use a smaller neighborhood than the previously introduced metaheuristics as a way to reduce the computation time. Instead of applying an exchange move on facilities, they first remove an open facility from the solution and then add a currently closed potential facility to the partial solution. They also parallelize multiple VNS procedures. Their PVNS metaheuristic overall outperforms the approaches of Belotti et al. (2007) and Colmenar et al. (2016).

Mladenović et al. (2020) develop a basic variable neighborhood search (VNS) for the OPMP that follows the "less is more" principle. They start from a random initial solution and apply a simple local search strategy in which, for a given facility, the best replacement facility is identified. Furthermore, they use a shaking step in which two facilities are exchanged randomly. The VNS is able to outperform the single-thread approaches from Belotti et al. (2007) and Colmenar et al. (2016) and derives solutions with a slightly worse solution quality than the PVNS of Herrán et al. (2020).

Gokalp (2020) propose an iterated greedy (IG) metaheuristic for the OPMP. The IG metaheuristic starts from a random initial solution and then applies a destruction and a construction phase. In the destruction phase, a given number of facilities are closed randomly. In the construction phase, a given number of facilities are opened using a greedy rule. The IG metaheuristic outperforms the approaches of Belotti et al. (2007), Colmenar et al. (2016), and Mladenović et al. (2020) and is competitive with the metaheuristic of Herrán et al. (2020).

Chang et al. (2021) introduce a parallel iterative solution-based tabu search (PISTS) metaheuristic for the OPMP. Similar to Herrán et al. (2020), they apply a remove-add move instead of an exchange move. Additionally, they track already visited solutions in a tabu list so that these solutions cannot be revisited. They parallelize their approach using multiple threads, as done in Herrán et al. (2020). The multiple threads share one tabu list. Their PISTS metaheuristic finds on average solutions with a better objective function value than the approaches of Belotti et al. (2007), Colmenar et al. (2016), Herrán et al. (2020), and Mladenović et al. (2020).

3.3.3 Research gap

Although there have been great advancements in exact approaches for the OPMP, they are still not scalable to medium and large instances. Metaheuristic approaches are generally known to find good solutions in short running times. However, they cannot easily be adapted to extensions of a planning problem. In this regard, the matheuristic we propose in Section 3.4 is more flexible in considering additional practical constraints. Another disadvantage of the metaheuristics from Section 3.3.2 is that they either exchange or remove and add facilities based on iterative procedures. Most of these metaheuristics only remove and add the best facility (in terms of solution quality) per iteration. Those that remove and add more than one facility per iteration apply a greedy rule to select the set of facilities. The solution that results from applying this greedy rule may not be the same as when the best set of facilities is removed and added. Additionally, the solution that results when the best facility is removed and added over multiple iterations may not be the same as when the best set of facilities is removed and added at once. We close this gap by introducing a matheuristic in which the best set of more than one facility can be removed and added in each iteration using mixed-integer programming.

3.4 Matheuristic

Matheuristics are recent approaches that exploit the mutual advantages of mathematical models and heuristic techniques (cf. Maniezzo et al., 2021). They combine the flexibility of mathematical models with the scalability of heuristic techniques. Additionally, the performance of mathematical programming solvers has improved considerably over the past few years (cf. Koch et al., 2022), which directly improves the performance of the matheuristics that are using these solvers. In this section, we present our matheuristic for the OPMP. In Section 3.4.1, we give an overview of our matheuristic. In Section 3.4.2, we describe the procedure to generate initial solutions. In Section 3.4.3, we describe the procedure to improve the initial solutions. We illustrate our matheuristic in Section 3.4.4 using the small example introduced in Section 3.2.2. Finally, in Section 3.4.5, we explain how the procedure of the matheuristic can be further sped up, especially for instances including many clients.

3.4.1 Overview

Our proposed matheuristic includes two procedures: an initialization procedure and an improvement procedure. In the initialization procedure, a diverse set of initial solutions

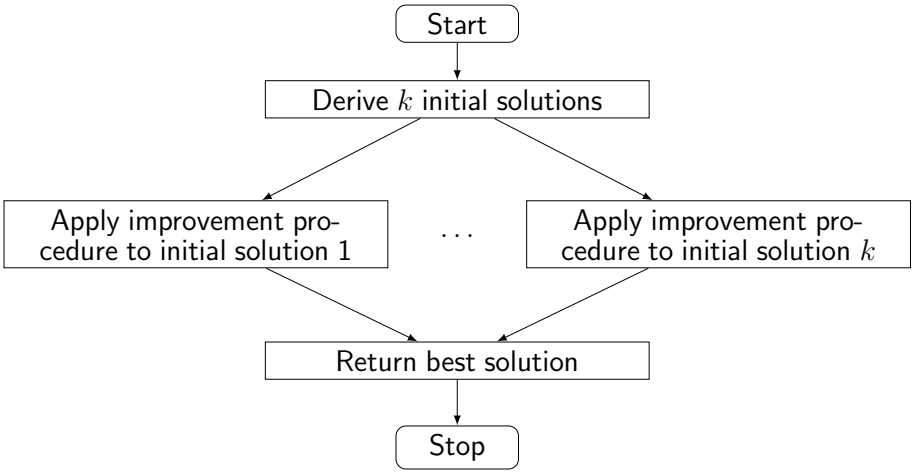

Figure 3.4.1: Multi-start of matheuristic

are constructed using heuristic techniques. In the improvement procedure, an initial solution is iteratively improved by applying a local search paired with mixed-integer programming. Our matheuristic is a multi-start approach where each initial solution is improved, but only the best solution is returned in the end. Figure 3.4.1 gives an overview of the multi-start approach of the matheuristic. Parameter k determines the number of initial solutions that are constructed and improved.

3.4.2 Generation of initial solutions

The leading metaheuristics generate the set of initial solutions independently of each other. However, when the initial solutions are generated independently from each other, they may be very similar to each other, which means that a local search starting from these different initial solutions explores very similar neighborhoods. Our main idea is to apply a procedure that guarantees the construction of initial solutions that are very different from each other, i.e., that cover different parts of the solution space. Consequently, the matheuristic searches a large portion of the solution space during the improvement iterations, which increases the likelihood of quickly finding high-quality solutions. We further observed that in good solutions to instances of the OPMP, open facilities are often located close to each other. Accordingly, a second rationale behind generating our initial solutions is to open facilities close to each other.

Assume that k initial solutions must be generated. We use the k-means++ algorithm to identify k facilities that are well distributed over the entire instance. From each of the k identified facilities, we generate an initial solution by opening the identified facility

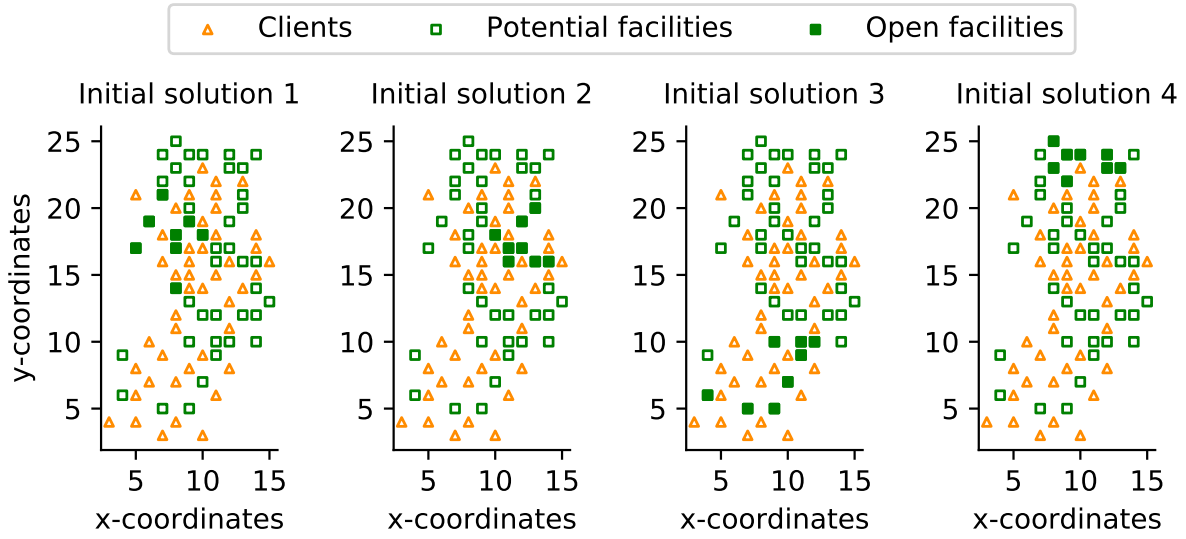

Figure 3.4.2: Four initial solutions for the illustrative example derived by the k-means++-based procedure described in Section 3.4.2

and its $p-1$ closest facilities. The p open facilities are stored in a set J^{sol}. Figure 3.4.2 visualizes for the illustrative example from Section 3.2.2 four different initial solutions that result from this procedure.

Furthermore, it can also be beneficial to include some randomly generated initial solutions. For example, if an instance includes two regions in which exactly $\frac{p}{2}$ potential facilities and no clients are located, a good solution would be to open the $\frac{p}{2}$ facilities in both regions. However, suppose the matheuristic starts from a solution in which all facilities are opened close to each other. In this case, it may not find the solution in which the $\frac{p}{2}$ facilities of both regions are opened. To help cover such special cases, in addition to the procedure from above, we decided to use two randomly generated initial solutions and $k-2$ initial solutions generated based on the above procedure.

Note that for some instances from the literature, the coordinates v_i, $i \in I$, of the clients and w_j, $j \in J$, of the facilities are not given. Only the distances d_{ij} between the clients and facilities are available for these instances. When we apply the matheuristic to these instances, the above procedure to generate initial solutions cannot be applied because the inter-facility distances are unavailable. Hence, we use an adapted version of the procedure. This adapted procedure is described in detail in Appendix 3.A.

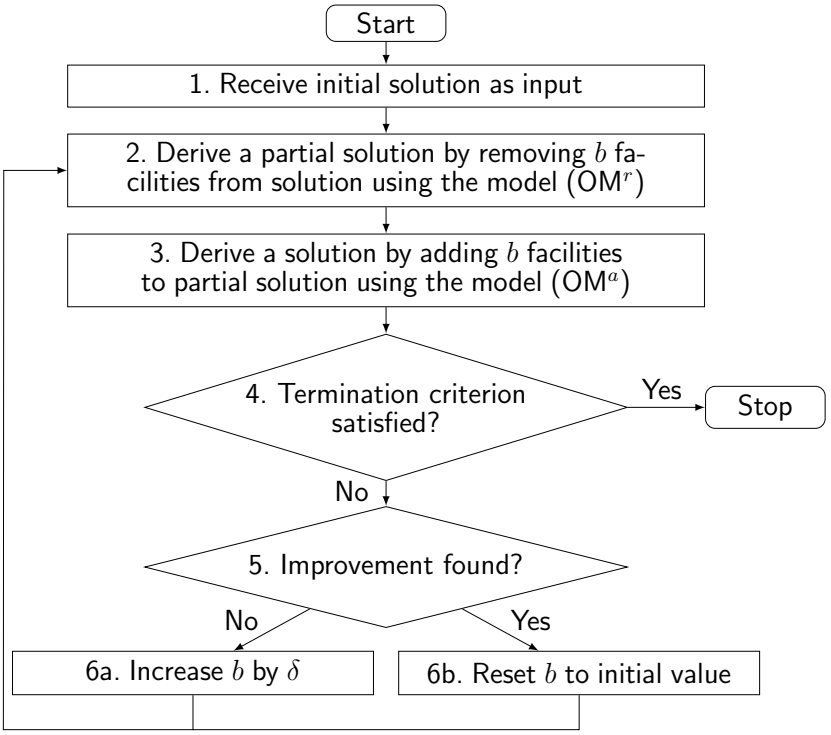

Figure 3.4.3: Flowchart of the improvement procedure of matheuristic

3.4.3 Improvement procedure

Flowchart 3.4.3 shows the different steps of the improvement procedure. In the following, we will describe these steps in more detail.

1. First, an initial solution in which p facilities are opened is received as input. We denote the set that contains the p currently opened facilities as J^{sol} (cf. Section 3.4.2).

2. Second, we remove a given number b of facilities from the current solution J^{sol}. A straightforward approach would be to use model (OM) from Section 3.2.1 to remove facilities from the solution. The variables y_j of the facilities $j \in J \setminus J^{sol}$ that are currently not opened could be fixed to a value of zero, and from the p currently opened facilities $j \in J^{sol}$, b would have to be closed. However, this model would contain many unnecessary variables and constraints. For large instances, it is no longer practicable to set up and solve such a model within an appropriate running time. Hence, we derive a version of model (OM), subsequently referred to as model (OM^r), that contains substantially fewer variables and constraints than model (OM) and can thus be set up and solved in a shorter running time. The main idea behind the adaptations in the model is that only facilities $j \in J^{sol}$ that are opened in the

Table 3.4.1: Additional notation of model (OM^r)

Set	Description
J^{sol}	Set of open facilities in current solution, i.e., $\{j \in J : y_j = 1\}$
$\dot{F}_i$	Facility sequence $\{\dot{f}_{ik} : k = 1, \ldots, b+1\}$ of facilities $j \in J^{sol}$ sorted in ascending order of the distances d_{ij} for each client i including facilities up to (and including) position $b+1$
$\dot{I}_j$	Set of clients $i \in I$ for which facility $j \in J^{sol}$ is among the b nearest facilities
$\dot{J}_i$	Set of facilities $j \in J^{sol}$ that are among the b nearest facilities for client $i \in I$
Parameter	
b	Number of facilities that are removed or added

current solution can be removed and have to be considered in the model (OM^r). Table 3.4.1 describes the additional notation used in the model (OM^r).

$$
(\mathrm{OM}^r)\begin{cases}
\text{Max. } (3.1) & & & \\
\text{s.t. } \displaystyle\sum_{j \in J^{sol}} y_j = p - b & & & (3.9)\\
D_i \le d_{i\dot{f}_{i,b+1}} - & & & \\
\qquad \displaystyle\sum_{k<b+1} (d_{i\dot{f}_{i,k+1}} - d_{i\dot{f}_{ik}}) v_{i\dot{f}_{ik}} & & (i \in I) & (3.10)\\
v_{i\dot{f}_{ik}} \le v_{i\dot{f}_{i,k+1}} & & (i \in I, k < b) & (3.11)\\
\displaystyle\sum_{i \in \dot{I}_j} v_{ij} \ge |\dot{I}_j| y_j & & (j \in J^{sol}) & (3.12)\\
y_j \in \{0,1\} & & (j \in J^{sol}) & (3.13)\\
v_{ij} \in \{0,1\} & & (i \in I, j \in \dot{J}_i) & (3.14)\\
D_i \in [0, d_{i\dot{f}_{i,b+1}}] & & (i \in I) & (3.15)
\end{cases}
$$

The objective function corresponds to the objective function of the model (OM). Constraint (3.9) ensures that exactly $p - b$ facilities are opened, i.e., that exactly b facilities are removed from the current solution J^{sol}. Constraints (3.10) pose an upper bound on the distance D_i from each client i to the client's nearest open facility.

Because exactly b facilities are removed, the distance D_i cannot exceed $d_{i\dot{f}_{i,b+1}}$, the distance to the facility that is at position $b+1$ of the facility sequence $\dot{F}_i$ of client i. Constraints (3.11) force the variable v_{ij} to take the value one if facility j is further away from client i than at least one open facility in J^{sol}. Constraints (3.12) ensure that the variables v_{ij} take the value one for each client $i \in \dot{I}_j$ if facility $j \in J^{sol}$ is opened. Finally, constraints (3.13), (3.14), and (3.15) specify the domains of the variables. To further speed up the solution process, we provide a warm start to the variables y_j. We sort the facilities $j \in J^{sol}$ in descending order of the sum of the distances $\sum_{i \in I} d_{ij}$ to all clients and provide a value of one as the initial value for the variables y_j of the first $p - b$ facilities. Then, the model (OM^r) is solved using a generic mixed-integer linear programming (MILP) solver. The $p - b$ facilities that are opened in the resulting partial solution are stored in a set J^{part}, i.e., $J^{part} = \{j \in J^{sol} : y_j = 1\}$.

3. Third, we add the same number b of facilities to the partial solution J^{part} from step 2. When adding facilities, we also substantially decrease the number of variables and constraints in the used model compared to the model (OM). These adjustments again considerably decrease the time used to set up and solve the model. The general idea is to only consider a subset of the potential facilities in each iteration. Hence, in each iteration, only facilities from a subset $J' \subseteq (J \setminus J^{part})$ of all potential facilities can be added. The parameter m^{max} denotes the number of potential facilities that can be considered in the model such that it can still be solved relatively quickly by a generic MILP solver. Hence, we set the value for parameter m^{max} depending on the number of clients in an instance (cf. Section 3.5.3). We describe in detail how we derive the subset of considered facilities J' in Appendix 3.B.

   Furthermore, we already know that the facilities of the partial solution $j \in J^{part}$ are open. Hence, the minimum distance of client i to these facilities poses an upper bound on the shortest distance D_i of client i to the nearest open facility. Thus, to derive the correct value of D_i, $i \in I$, only the facilities $j \in J'$ that are closer to client i than the closest facility $j \in J^{part}$ must be considered. In more detail, we can derive an upper bound u_i on the distance D_i to the nearest open facility of client i. This upper bound u_i is equal to the minimum distance of client i to the facilities $j \in J^{part}$, i.e., $u_i = \min_{j \in J^{part}}\{d_{ij}\}$, if $J^{part} \neq \emptyset$. If $J^{part} = \emptyset$, i.e., if all facilities are removed in step 2, u_i is set to $d_{i,f_{i,m-b+1}}$, the distance to the b-furthest facility from client i. Additionally, the variables v_{ij} are only required for each client $i \in I$ and the facilities $j \in J'$ for which $d_{ij} < u_i$. These adjustments again decrease the size of the model. We subsequently refer to the model used to add facilities to a partial

solution as the model (OM^a). Table 3.4.2 describes the additional notation used.

$$
(\mathrm{OM}^a)\begin{cases}
\text{Max. } (3.1) & & \\
\text{s.t. } \displaystyle\sum_{j\in J'} y_j = b & & (3.16)\\
D_i \le u_i - (u_i - d_{if'_{i,e_i}})v_{if'_{i,e_i}} & & \\
\quad - \displaystyle\sum_{k<e_i}(d_{if'_{i,k+1}} - d_{if'_{ik}})v_{if'_{ik}} & (i\in I : e_i > 0) & (3.17)\\
v_{if'_{ik}} \le v_{if'_{i,k+1}} & (i\in I, k < e_i) & (3.18)\\
\displaystyle\sum_{i\in I} v_{ij} \ge s_j y_j & (j\in J') & (3.19)\\
y_j \in \{0,1\} & (j\in J') & (3.20)\\
v_{ij} \in \{0,1\} & (i\in I, j\in J' : & \\
 & \quad d_{ij} < u_i) & (3.21)\\
D_i \in [0, u_i] & (i\in I) & (3.22)
\end{cases}
$$

The objective function is the same as in models (OM) and (OM^r). Constraint (3.16) ensures that exactly b facilities are opened in addition to the $p-b$ facilities in J^{part} that are already open. Constraints (3.17) pose an upper bound on the distance D_i from each client i to the client's nearest open facility. The right-hand sides of constraints (3.17) result from the upper bound u_i and the distances d_{ij} of client i to the considered facilities $j \in J' : d_{ij} < u_i$. Constraints (3.18) force the variable v_{ij} to take the value one if facility j is further away from client i than at least one open facility $j \in J' : d_{ij} < u_i$. Constraints (3.19) ensure that the variables v_{ij} take the value one for each client $i \in I$ if facility $j \in J'$ is opened. Finally, constraints (3.20), (3.21), and (3.22) define the domains of the variables. We again provide a warm start to the variables y_j to speed up the solution process. We provide an initial value of one to the variables y_j, $j \in J^{sol} \setminus J^{part}$, i.e., of the facilities that were removed in step 2. Next, the model (OM^a) is solved using a generic MILP solver. The set J^{sol} is updated to contain the facilities that are opened in the resulting solution, i.e., $J^{sol} = J^{part} \cup \{j \in J' : y_j = 1\}$.

4. Fourth, we check if a termination criterion is satisfied. We use a time limit combined with the condition $b = p$ as the termination criteria. As long as neither of the two termination criteria are satisfied, we go to step 5; otherwise, we stop.

5. Fifth, we check if the solution derived in step 3 is an improvement over the previous

Table 3.4.2: Additional notation of model (OM^a)

Set	Description
J^{part}	Set of open facilities in current partial solution, i.e., $\{j \in J^{sol} : y_j = 1\}$
J'	Set of facilities that are considered in current iteration
F'_i	Facility sequence $\{f'_{ik} : k = 1, \ldots, e_i\}$ of facilities $j \in J' : d_{ij} < u_i$ sorted in ascending order of the distances d_{ij} for each client i
Parameter	
e_i	Number of facilities $j \in J'$ with $d_{ij} < u_i$, $i \in I$
s_j	Number of clients $i \in I$ with $d_{ij} < u_i$, $j \in J'$
u_i	$= d_{i,f_{m-b+1}}$, if $J^{part} = \emptyset$; $= \min_{j \in J^{part}}\{d_{ij}\}$, otherwise

solution, i.e., if the objective function value of the solution derived in step 3 is higher than the objective function value of the solution derived in the last iteration (or the initial solution if we are in the first iteration). If no improvement has been found, we go to step 6a. Otherwise, we go to step 6b.

6a. In step 6a, we increase b by a given number denoted by δ.

6b. In step 6b, we reset b to its initial value.

3.4.4 Illustrative example

We illustrate the matheuristic using the small example from Section 3.2.2 with a single-start procedure, i.e., we only consider one initial solution. We set the parameters $b = 2$, $\delta = 1$, and $m^{max} = 20$, and we set a time limit of 10 seconds and the condition $b = p$ as the termination criteria. Figure 3.4.4 illustrates the first two iterations.

First, we generate an initial solution. The potential facility with coordinates $(12, 17)$ is arbitrarily opened here. Then, the $p - 1 = 7$ nearest facilities are additionally opened, which results in the initial solution depicted in Figure 3.4.4 (Iteration 1, left).

Second, we derive a partial solution, i.e., we remove two facilities from the initial solution by applying the model (OM^r). Here, the two facilities with coordinates $(10, 18)$ and $(11, 16)$ are removed.

Third, we derive a solution, i.e., we add two facilities to the partial solution from step 2 by applying the model (OM^a). The set J' of the considered facilities is highlighted

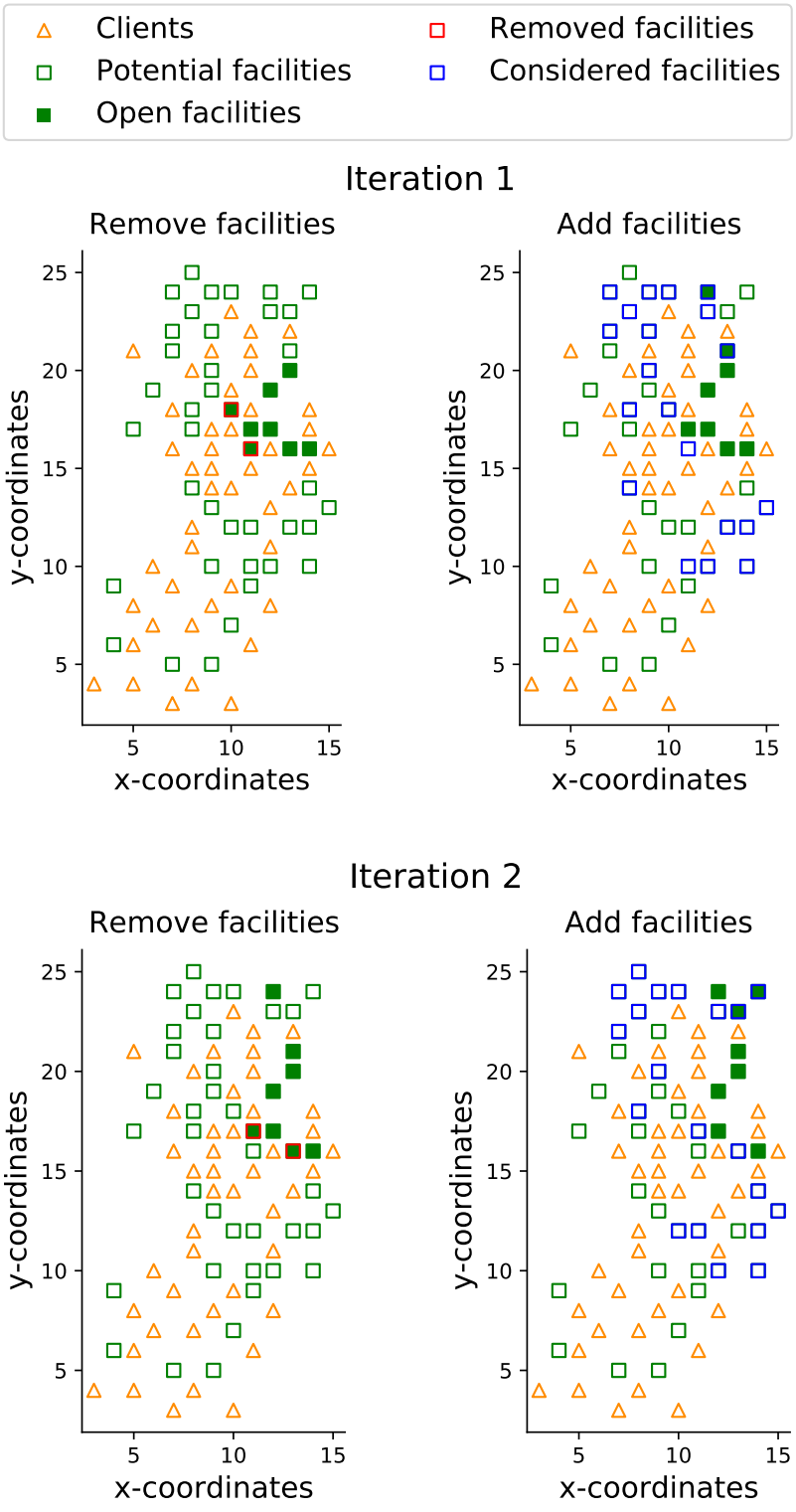

Figure 3.4.4: Iterations 1 and 2 of the matheuristic for the illustrative example

in blue. It contains $m^{max} = 20$ facilities that are selected based on the distance-based priority rule described in Appendix 3.B. Here, the two facilities with coordinates $(13, 23)$ and $(14, 24)$ are opened. They are marked in solid green with a blue border in Figure 3.4.4 (Iteration 1, right).

Fourth, we check if the time limit is reached or if $b = p$. Because neither are satisfied, we go to step 5.

Fifth, we check if an improvement has been found. The objective function value of the initial solution is 233.66, and the objective function value of the solution derived in Iteration 1 is 251.74. Hence, an improvement has been found, and we go to step 6b.

In step 6b, parameter b is reset to its initial value of two.

In Iteration 2, again, two facilities are removed from the current solution. The two facilities are highlighted in red in Figure 3.4.4 (Iteration 2, left). Figure 3.4.4 (Iteration 2, right) shows the two facilities that are added to the partial solution from Figure 3.4.4 (Iteration 2, left). Again, an improvement is found. Figures 3.C.1–3.C.5 in Appendix 3.C illustrate the remaining iterations. The matheuristic stops after Iteration 12 because $b = p$, i.e., all facilities are removed and added again.

3.4.5 Clustering of clients

For large instances involving thousands of clients, we additionally propose a scaling technique. The idea is to decrease the number of clients that must be considered by clustering them. By considering fewer clients (or clusters of clients), the number of variables and constraints in the models (OM^r) and (OM^a) is further decreased, which leads to speed-ups in the solution processes of the models in each iteration. On the other hand, an aggregated version of the distance matrix must be used, which provides less accurate information on the impact of opening a specific facility. To cluster clients, we must decide on the clustering algorithm, the number of clusters, and how to determine the distances between the clusters of clients and potential facilities.

We use the k-means algorithm to cluster clients because its running time remains low, even if the number of objects is large. It is essential to cluster clients quickly because we want to use most of the available time in the improvement procedure of our matheuristic. We derive the number of clusters from a percentage r and the number of clients n in the instance as $\lfloor r \times n \rfloor$ (cf. Section 3.5.6). Let C denote the set that contains the cluster indices, i.e., $C = \{1, \ldots, \lfloor r \times n \rfloor\}$, and C_l denote the set that contains all clients of cluster $l \in C$. Finally, we set the distance d_{lj} of cluster l to potential facility j to the sum of the distances of all clients in cluster l to the potential facility j, i.e., $d_{lj} = \sum_{i \in C_l} d_{ij}$, $l \in C, j \in J$. Subsequently, we refer to the distances d_{lj}, $l \in C, j \in J$ as the adjusted

distances.

After clustering the clients into $\lfloor r \times n \rfloor$ clusters, we apply the matheuristic using the adjusted distances d_{lj}. Steps 1–6 of the improvement procedure work as described in Section 3.4.3 except for some minor adjustments that must be made to the models (OM^r) and (OM^a). In the models (OM^r) and (OM^a), all sets related to the clients $i \in I$ must be replaced by sets associated with the clusters $l \in C$. This means that the clusters $l \in C$ are treated as the clients. Furthermore, the distances d_{ij}, $i \in I, j \in J$ must be replaced by the adjusted distances $d_{lj}, l \in C, j \in J$. Hence, the facility sequences $\dot{F}_l$ and F'_l are derived for each cluster $l \in C$ based on the adjusted distances d_{lj}. After a termination criterion is satisfied, the solution quality must be evaluated based on the original distances d_{ij}, $i \in I, j \in J$. Figure 3.4.5 (left) illustrates for the illustrative example the clusters that result when deriving $|C| = \lfloor r \times n \rfloor = 0.2 \times 45 = 9$ clusters. The objective function value of the solution depicted in Figure 3.4.5 (right) using the adjusted distances d_{lj} is 460.0. When using the original distances d_{ij}, the objective function value is 455.7, as in the solution illustrated in Figure 3.2.1. Note that the difference between these two values arises because not all clients of a cluster are nearest to the same open facility when the original distances d_{ij} are used.

3.5 Computational results

In this section, we compare the performance of our matheuristic to the performance of the two leading metaheuristics from the literature. In Section 3.5.1, we describe the problem instances used for the computational experiment. In Section 3.5.2, we provide details on the two metaheuristics from the literature. In Section 3.5.3, we present the experimental design and explain how we derived the parameter settings used in Sections 3.5.4 and 3.5.5. In Section 3.5.4, we compare the performance of the matheuristic to the performances of the two leading metaheuristics for small benchmark instances from the literature. In Section 3.5.5, we compare their performances for newly generated instances, including medium and large instances. Finally, in Section 3.5.6, we show how the performance of the matheuristic can be further improved by clustering the clients.

3.5.1 Instances

We use two test sets to compare the performance of the matheuristic to the performances of the two leading metaheuristics. The first test set comprises 144 small test instances that were first introduced by Colmenar et al. (2016) for the OPMP (cf. `https://grafo.etsii.urjc.es/optsicom/opm.html`). These instances include up to $n = 450$ clients and

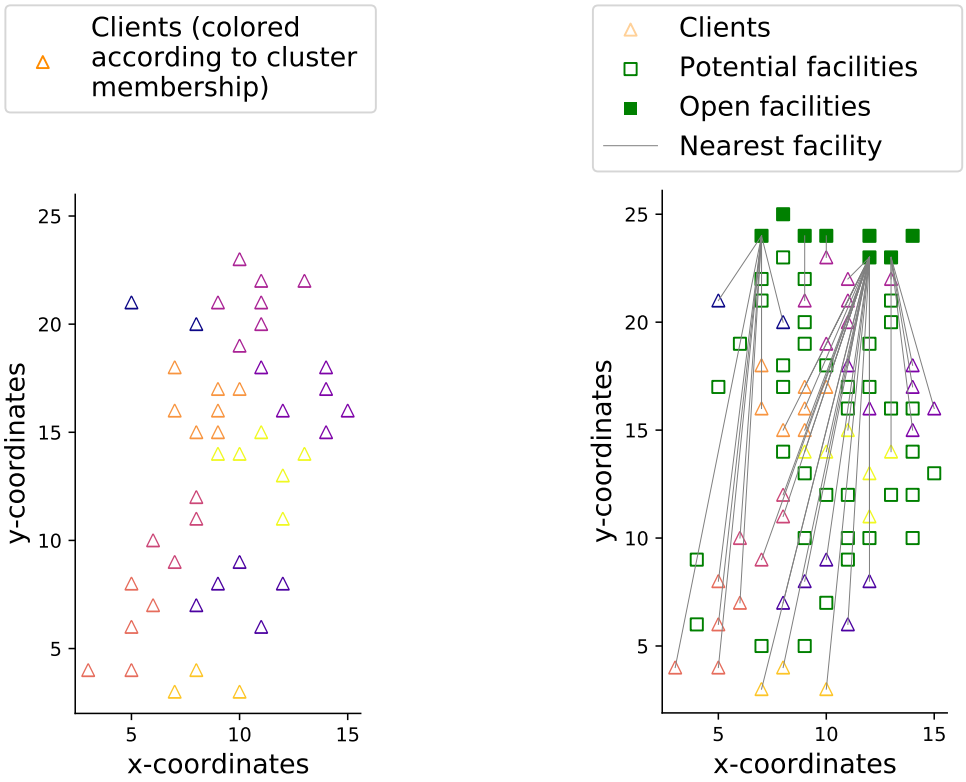

Figure 3.4.5: Cluster membership of the clients (left) and solution derived by the matheuristic when the clients are clustered (right) with $r = 0.2$

$m = 450$ potential facilities, and the value for p is set as $\frac{m}{8}$ (low), $\frac{m}{4}$ (medium), and $\frac{m}{2}$ (high). We subsequently refer to the instances of this test set as the benchmark instances.

The second test set consists of various small, medium, and large test instances that we derived from instances of the VLSI collection (cf. Rohe, 2013) and the TSPLIB (cf. Reinelt, 1991). We used the same procedure as described in Colmenar et al. (2016), i.e., we randomly split the original data points into a set of clients and a set of potential facilities, and we set the value of p to $\frac{m}{8}$ (low), $\frac{m}{4}$ (medium), and $\frac{m}{2}$ (high). Table 3.D.1 in Appendix 3.D reports the main characteristics of the instances in this test set. The instances include up to $n + m = 1{,}972$ clients and potential facilities (small), up to $n + m = 15{,}112$ clients and potential facilities (medium), and up to $n + m = 21{,}214$ clients and potential facilities (large). Other than Colmenar et al. (2016), we varied the percentage of the original data points that we selected as clients and potential facilities. Instances 1–36 and 55–69 include the same number of clients and potential facilities, i.e., $n = m$, as in Colmenar et al. (2016). Instances 37–45 comprise 75% clients and 25% potential facilities, i.e., $n > m$, and instances 46-54 comprise 25% clients and 75%

potential facilities, i.e., $n < m$. The shape of the instances is either "square", where all data points lie within a rectangular shape or "other", where the shape of the instance is arbitrary. We subsequently refer to the instances of this test set as the new instances. We have also made these test instances available to the community (cf. `https://github.com/tabigler/OPMP-instances`).

3.5.2 Leading metaheuristics

We compare our matheuristic to the two leading metaheuristics PISTS and IG of Chang et al. (2021) and Gokalp (2020), respectively. For the first time, the performances of the PISTS and IG metaheuristics are also directly compared. These two metaheuristics both start from randomly generated initial solutions. For both metaheuristics, we use the parameter settings that were used in the respective paper. We check for each instance the time required by the PISTS metaheuristic, which results from a time limit of 2,834 seconds and an iteration limit of $20p$ (cf. Chang et al., 2021). The time required by the PISTS metaheuristic is set as a time limit for the matheuristic and the IG metaheuristic. We apply a multi-start with eight different initial solutions for the PISTS and the IG metaheuristic (analogously to the matheuristic, cf. Section 3.5.3).

3.5.3 Experimental design

We tested our matheuristic and the PISTS and IG metaheuristics on an HP workstation with one Intel Xeon CPU with 2.20 GHz clock speed and 128 GB RAM. The matheuristic was implemented in Python 3.7, and we used the Gurobi Optimizer 9.5.2 as a solver. The code of the matheuristic is available upon request. Note that all distances computed are Euclidean distances and rounded to the nearest integer. We applied the matheuristic with a multi-start with eight different initial solutions. Two of the eight initial solutions are set to random initial solutions for both test sets. The rest of the initial solutions are generated according to the procedures described in Section 3.4.2.

We considered three different values for each of the parameters b, δ, and m^{max} and tested all combinations of these parameter values on a small, representative subset of the instances. The considered values are the following: $b = 1, 5, 10$, $\delta = 1, 2, 5$, and $m^{max} = \frac{300^2}{n}, \frac{450^2}{n}, \frac{600^2}{n}$. The value of parameter m^{max} depends on the number of clients n because the number of variables in the model (OM^a) increases with n and m. If we want to keep the models small, fewer potential facilities can be considered if an instance includes many clients, and vice versa. In preliminary tests, the model (OM^a) could be solved in a reasonable running time when including approximately 450 clients and 450

potential facilities. Thus, we tested values for m^{max} in a similar range. For the benchmark instances, we used the same subset of instances as in Colmenar et al. (2016) to tune the parameters. For the new instances, we used a subset of seven representative instances to tune the parameters. We selected the instances such that different instance sizes and values of p are represented. These instances are highlighted in bold in Table 3.D.1. Please note that for the instances of Colmenar et al. (2016), the value of $m^{max} = \frac{600^2}{n}$ was not tested because with the value $m^{max} = \frac{450^2}{n}$ all potential facilities are already considered in each iteration, i.e., $J' = J$. We selected the best combination of parameter values concerning the average objective function value for both test sets. For the benchmark instances, several combinations resulted in the same average objective function value. Thus, we selected the combination for which the matheuristic derived the best solutions on average at the earliest time, which resulted in the values $b = 1$, $\delta = 2$, and $m^{max} = \frac{450^2}{n}$. For the new instances, the values $b = 5$, $\delta = 1$, and $m^{max} = \frac{450^2}{n}$ yielded the best solution quality on average. Note that, however, the differences in the average objective function values for the different values of m^{max} and δ are rather small. Because we only use a small subset of instances for deriving the parameter values, these instances are also included in the results in the next two sections. Note that the main insights would not change if these instances were not included in the results.

3.5.4 Results benchmark instances

In this section, we compare the performance of the matheuristic to the performances of the two leading metaheuristics based on the small benchmark instances from Section 3.5.1. Figure 3.5.1 shows the average gap between the solutions derived by the matheuristic and the solutions derived by the PISTS and IG metaheuristics. The average gap is zero for the combinations of n, m, and p, where no bar is visible in the plot. On average, the matheuristic derives slightly better solutions in terms of solution quality than the PISTS metaheuristic, with an average gap of -0.011%. The average gap to the solutions derived by the IG metaheuristic on the benchmark instances is slightly positive (0.003%), meaning that our matheuristic derives solutions with a slightly inferior solution quality on average. However, due to the small difference, we can conclude that the results are competitive for the benchmark instances. Table 3.E.1 in Appendix 3.E contains the objective function values derived by the matheuristic, the PISTS metaheuristic, and the IG metaheuristic for all benchmark instances.

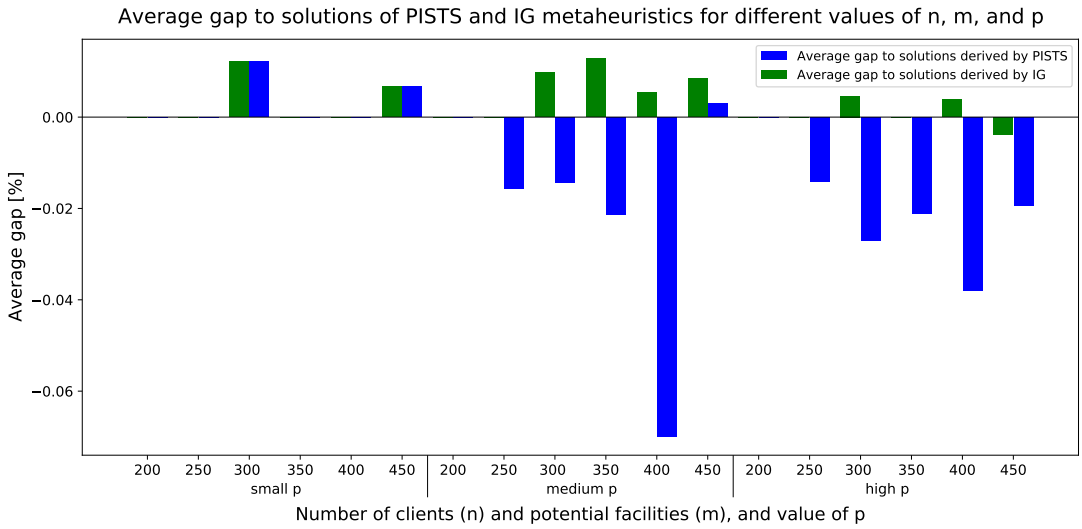

Figure 3.5.1: Results for the benchmark instances (a negative average gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

3.5.5 Results new instances

In this section, we compare the performance of the matheuristic to the performances of the two leading metaheuristics based on the new instances (cf. Section 3.5.1). Table 3.5.1 reports the average gaps of the solutions derived by the matheuristic to the solutions derived by the PISTS metaheuristic (∅ $\text{Gap}^{\text{PISTS}}$) and the IG metaheuristic (∅ Gap^{IG}) for instances 1–36 with $n = m$ and a squared shape. Table 3.5.2 reports the gaps ($\text{Gap}^{\text{PISTS}}$ and Gap^{IG}) between these approaches for instances 37–54 with $n \neq m$ and a squared shape. Table 3.5.3 reports the average gaps between the three approaches for instances 55-69, which have an arbitrary shape. The results in the tables are grouped by the size of the instances (small, medium, large) and the value of p (low, medium, high). The matheuristic is competitive with the leading metaheuristics for small instances. For medium instances, it clearly outperforms the PISTS metaheuristic in terms of solution quality. It derives very similar results to the IG metaheuristic for medium instances with low and medium values of p. For medium-sized instances with a high value of p, it outperforms the IG metaheuristic. For large instances, the matheuristic substantially outperforms both metaheuristics in terms of solution quality. Table 3.E.2 in Appendix 3.E contains the objective function values derived by the three approaches for all new instances.

We also tested the effect of our distance-based priority rule to derive the considered facilities J'. We compared the distance-based priority rule to a random selection of the considered facilities J'. Using our distance-based priority rule improves the average gap of

Table 3.5.1: Aggregated results for instances with shape "squared" and $n = m$ (a negative average gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

	∅ $\text{Gap}^{\text{PISTS}}$ [%]			∅ Gap^{IG} [%]		
Size	low p	medium p	high p	low p	medium p	high p
small	0.1	0.0	-1.0	0.1	-0.4	-0.2
medium	-1,102.3	-1,554.8	-1,225.2	0.3	0.1	-6.4
large	-3,235.8	-3,369.1	-2,271.4	-153.1	-177.2	-146.8

Table 3.5.2: Results for instances with shape "squared" and $n \neq m$ (a negative gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

		$\text{Gap}^{\text{PISTS}}$ [%]			Gap^{IG} [%]		
Division	Size	low p	medium p	high p	low p	medium p	high p
$n < m$	small	-7.0	0.0	0.0	0.0	0.0	-15.6
	medium	-2,421.5	-3,017.5	-2,238.1	0.1	0.0	-0.2
	large	-3,383.8	-3,324.4	-2,043.2	-223.4	-243.7	-192.6
$n > m$	small	0.0	0.0	-9.8	0.0	0.0	-9.8
	medium	-239.7	-1,002.3	-960.0	4.1	0.1	-4.9
	large	-1,761.7	-1,777.7	-1,192.1	-2.5	-28.5	11.0

Table 3.5.3: Aggregated results for instances with shape "other" (a negative average gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

	∅ $\text{Gap}^{\text{PISTS}}$ [%]			∅ Gap^{IG} [%]		
Size	low p	medium p	high p	low p	medium p	high p
small	0.1	-3.9	0.0	0.1	0.0	0.0
medium	-1,914.0	-2,066.4	-1,596.1	1.3	1.8	-0.4

all new instances to the solutions derived by the PISTS metaheuristic by 72.23 percentage points and to the solutions derived by the IG metaheuristics by 4.46 percentage points compared to using a random selection. Using our distance-based priority rule also yields better results than applying a deterministic greedy selection of the facilities where the nearest facilities to the currently opened facilities are selected.

From Tables 3.5.1 and 3.5.2, we can see that our matheuristic performs especially well for instances for which the number of potential facilities m is the same or larger than the number of clients n. The inferior performance on instances with $n > m$ motivates the clustering of clients introduced in Section 3.4.5. We discuss the results of the matheuristic with the clustering of clients in the following section.

3.5.6 Results when clustering clients

In this section, we show how the clustering of clients further improves the performance of the matheuristic, especially for large instances with $n > m$. First, we explain how we determined a value for the parameter r, which represents a percentage that, when multiplied by the number of clients, results in the number of clusters that are created. Then, we present the results of the matheuristic when clients are clustered for the medium and large instances of the second test set (new instances).

We ran the matheuristic for different values of r (1%, 5%, and 10%) on the medium and large new instances. We omitted the small instances because the clustering of clients is only useful when the number of clients is large. On average, the matheuristic with $r = 5\%$ derived the best solution quality. Figure 3.5.2 indicates for the three exemplary instances 64–66 how the objective function value changes over time for the matheuristic and the matheuristic with clustering of clients with $r = 1\%$, 5%, and 10%. The three

Table 3.5.4: Aggregated results for instances with shape "squared" and $n = m$ when clients are clustered with $r = 5\%$ (a negative average gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

	∅ $\text{Gap}^{\text{PISTS}}$ [%]			∅ Gap^{IG} [%]		
Size	low p	medium p	high p	low p	medium p	high p
medium	-1,113.4	-1,562.6	-1,228.9	-0.1	-0.2	-6.7
large	-3,281.6	-3,546.1	-2,351.6	-155.9	-199.2	-158.3

instances 64–66 are based on the same clients and potential facilities and differ only by their value of p (low, medium, high). Through the clustering of clients, the number of clients (clusters) that must be considered in the models (OM^r) and (OM^a) is smaller, which decreases the model sizes and running times to solve the models. We can see from Figure 3.5.2 that the iterations take less time when the clients are clustered. Additionally, the value of m^{max} depends on n; thus, when fewer clients (clusters) must be considered in the model (OM^a), m^{max} is set to a larger value. Hence, more potential facilities J' are considered in each iteration, and overall, the improvements found are larger. From Figure 3.5.2, we can also see that the effect of clustering clients is larger when the value of p is medium or high than when it is low. We also applied an elbow heuristic with $r = 1\%$, 5%, 10%, 15%, and 20% to the representative instance 64. Hence, we clustered the clients using different values of r and visualized the sum of the squared Euclidean distances of all clients to their nearest cluster center (objective function) for the different values of r (cf. Figure 3.5.3). We can see that a value of $r = 5\%$ seems appropriate (for this instance) because increasing r from 1% to 5% leads to a substantial decrease in the objective function value, while further decreasing r only has a small effect. Note that the plot looks similar for other instances of this test set.

Tables 3.5.4, 3.5.5, and 3.5.6 show the (aggregated) results of the matheuristic with $r = 5\%$ on all medium and large instances. From these tables, we can see that the clustering of clients leads to a substantial improvement in the results, especially when $n > m$, $n = m$, or when the value of p is high. This framework of clustering clients and applying an optimization approach to the reduced-size instance could also be helpful for other heuristic approaches or mixed-integer linear programs for the obnoxious p-median problem.

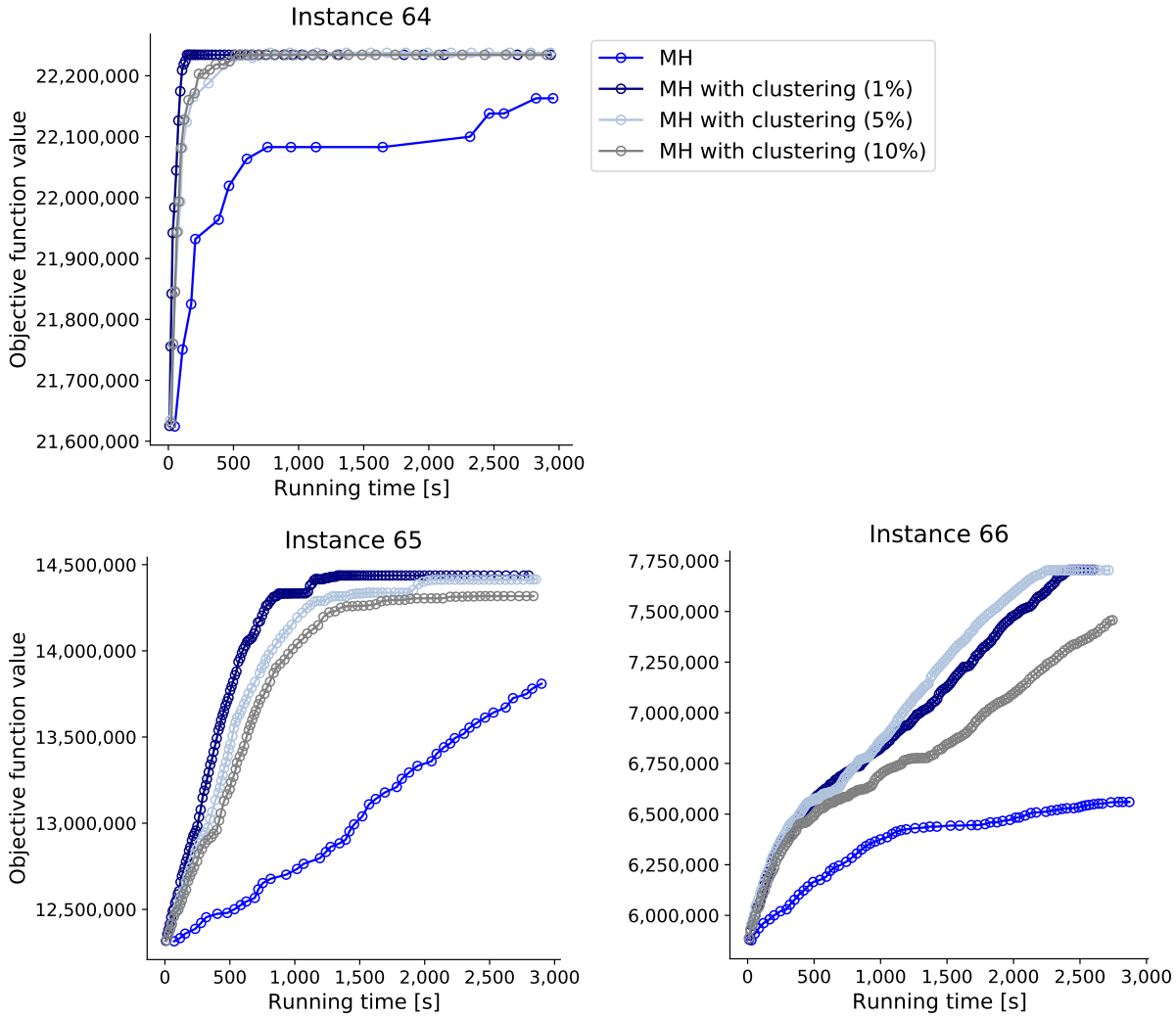

Figure 3.5.2: Objective function value over time for the matheuristic without and with the clustering of clients for instances 64–66 (note that higher values are better)

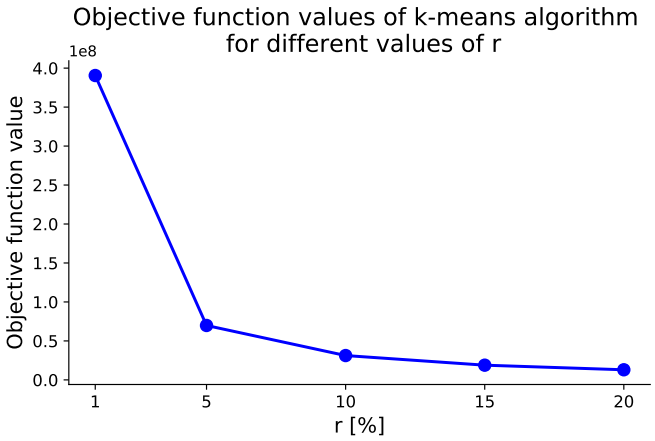

Figure 3.5.3: Elbow heuristic for selected instance 64 (note that smaller values are better)

Table 3.5.5: Results for instances with shape "squared" and $n \neq m$ when clients are clustered with $r = 5\%$ (a negative gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

		$\text{Gap}^{\text{PISTS}}$ [%]			Gap^{IG} [%]		
Division	Size	low p	medium p	high p	low p	medium p	high p
$n < m$	medium	-2,427.4	-3,027.8	-2,237.1	-0.2	-0.3	-0.1
	large	-3,432.1	-3,330.0	-2,273.8	-227.9	-244.2	-224.1
$n > m$	medium	-254.4	-1,003.1	-960.0	0.0	0.0	-4.9
	large	-1,828.7	-2,060.8	-1,377.0	-6.2	-47.9	-1.8

Table 3.5.6: Aggregated results for instances with shape "other" when clients are clustered with $r = 5\%$ (a negative average gap indicates that the matheuristic outperforms the metaheuristics in terms of solution quality)

	∅ $\text{Gap}^{\text{PISTS}}$ [%]			∅ Gap^{IG} [%]		
Size	low p	medium p	high p	low p	medium p	high p
medium	-1,986.3	-2,185.5	-1,656.9	-1.4	-2.1	-3.1

3.6 Extension

One great advantage of matheuristics over metaheuristics is that practical constraints can easily be incorporated into a matheuristic. In this section, we present an extension of the OPMP that includes such practical constraints. In Section 3.6.1, we introduce the extension and describe the few minor changes required in the matheuristic to be applicable to this extension of the OPMP. In Section 3.6.2, we analyze how the adjustments affect the structure of the solutions.

3.6.1 Description of extension

We observe that solutions to the OPMP are often clustered, i.e., the open facilities are all located very close to one another. Figure 3.2.1 illustrates this observation for the illustrative example. In real-world applications, however, such clustered solutions may not be ideal, and for political reasons, authorities may want to include practical constraints to control the placement of obnoxious facilities. Many constraints are plausible; in this paper, we concentrate on practical constraints that are set up for different regions of an instance. In real-world instances, regions are often given as neighborhoods in a city or states in a country. We assume that the authorities want to ensure that not all facilities are opened in the same region. Hence, the extension includes upper bounds on the number of open facilities per region of an instance.

The model (OM) of Lin and Chiang (2021) can be extended to include these practical constraints as follows. In addition to the already introduced notation, let H denote the set of all regions. The parameters r_h indicate the maximum number of open facilities per region $h \in H$ and the sets J_h denote the sets of facilities $j \in J$ that lie in the region $h \in H$. Constraints (3.23) ensure the upper bound r_h on the number of open facilities for each region $h \in H$.

$$\sum_{j \in J_h} y_j \leq r_h \qquad (h \in H) \tag{3.23}$$

The model (OM^{ext}) can be used to derive a solution to the extension of the OPMP and reads as follows.

$$(\text{OM}^{ext}) \begin{cases} \text{Max. } (3.1) \\ \quad \text{s.t. } (3.2) - (3.7) \\ \qquad (3.23) \end{cases}$$

Two minor changes to the matheuristic are required to derive feasible solutions to an

instance of the extension of the OPMP:

1. The generated initial solution must be feasible, i.e., satisfy the upper bounds on the number of open facilities per region.

2. When facilities are added to the partial solution in step 3 of the matheuristic, the model (OM^a) must include the upper bounds for each region to ensure feasibility.

We also consider additional constraints when removing facilities. If no such constraints are considered, the matheuristic often removes b facilities from one region and then adds the same b facilities again because the upper bounds on the number of open facilities in the other regions are tight. Hence, the matheuristic cannot open an additional facility in another region without violating one of the respective regional constraints. The matheuristic can often only escape this local optimum by substantially increasing the parameter b. Thus, we decided to include an additional constraint in the model (OM^r) that ensures that at least one facility is removed from each region in each iteration.

Next, we describe in detail how we generate the initial solutions and how we adapt the models (OM^r) and (OM^a). We derive an initial solution for an instance of the extension as follows. In our instances, $\sum_{h \in H} r_h = p$; thus, we randomly select exactly r_h facilities from each region $h \in H$ and open them. If $\sum_{h \in H} r_h$ is greater than p, we can derive an initial solution by opening fewer than r_h facilities in each of the regions in such a way that the number of opened facilities sums up to p. Let H^* denote a set that is set to H if $b \geq |H|$, i.e., if more or the same number of facilities must be removed than there are regions. If $b < |H|$, i.e., if fewer facilities must be removed than there are regions, b regions are selected randomly from H into H^*. We add the constraints (3.24) to the model (OM^r) to ensure that at least one facility must be removed from each region $h \in H^*$.

$$\sum_{j \in J^{sol} \cap J_h} y_j \leq |J^{sol} \cap J_h| - 1 \qquad (h \in H^*) \tag{3.24}$$

The adapted model ($\text{OM}^{r\ ext}$) reads as follows.

$$(\text{OM}^{r\ ext}) \begin{cases} \text{Max. } (3.1) \\ \quad \text{s.t. } (3.9) - (3.15) \\ \qquad (3.24) \end{cases}$$

Furthermore, we add the constraints (3.25) to the model (OM^a) to ensure that the

upper bounds r_h on the number of open facilities per region $h \in H$ are not exceeded.

$$\sum_{j \in J' \cap J_h} y_j \leq r_h - |J^{part} \cap J_h| \qquad (h \in H) \tag{3.25}$$

The adapted model (OM$^{a\ ext}$) reads as follows.

$$(\text{OM}^{a\ ext}) \begin{cases} \text{Max. } (3.1) \\ \quad \text{s.t. } (3.16) - (3.22) \\ \qquad\quad (3.25) \end{cases}$$

3.6.2 Results on selected instances

In this section, we first give some information on the instances used below and then present the results of the matheuristic for two exemplary instances.

In our instances, the parameters r_h were generated so that they are inversely proportional to the number of clients in a region and sum up to p. If there are many clients in a region, only a few facilities can be opened in this region, and vice versa. The assigned region per client and potential facility and the parameters r_h per region are available at `https://github.com/tabigler/OPMP-instances` for the instances used below.

First, we look at instance 59. For this instance, we generated four regions $H = \{1, 2, 3, 4\}$ with $r_1 = 33$, $r_2 = 46$, $r_3 = 44$, and $r_4 = 49$. When the matheuristic is applied to instance 59 without the practical constraints using a time limit of 500 seconds, it stops in the solution depicted in Figure 3.6.1 (left). In this solution, almost all facilities are opened in region 2. However, if we consider the extension of the OPMP, where the upper bounds on the number of open facilities per region are included, the matheuristic stops (after the same time limit) in the solution depicted in Figure 3.6.1 (right). Clearly, the facilities are better divided among the four regions in Figure 3.6.1 (right) than in Figure 3.6.1 (left).

Second, we examine instance 61. For this instance, we generated three regions $H = \{1, 2, 3\}$ with $r_1 = 88$, $r_2 = 102$, and $r_3 = 88$. Figure 3.6.2 (left) shows the solution that the matheuristic derives for the OPMP without the practical constraints using a time limit of 500 seconds. Again, all facilities are opened in the same region in this solution. Figure 3.6.2 (right) shows the solution that the matheuristic derives for the extension of the OPMP, where the facilities are opened in the three different regions. We also applied the matheuristic to other instances including the upper bounds per region and comprising thousands of clients and potential facilities and observed similar results.

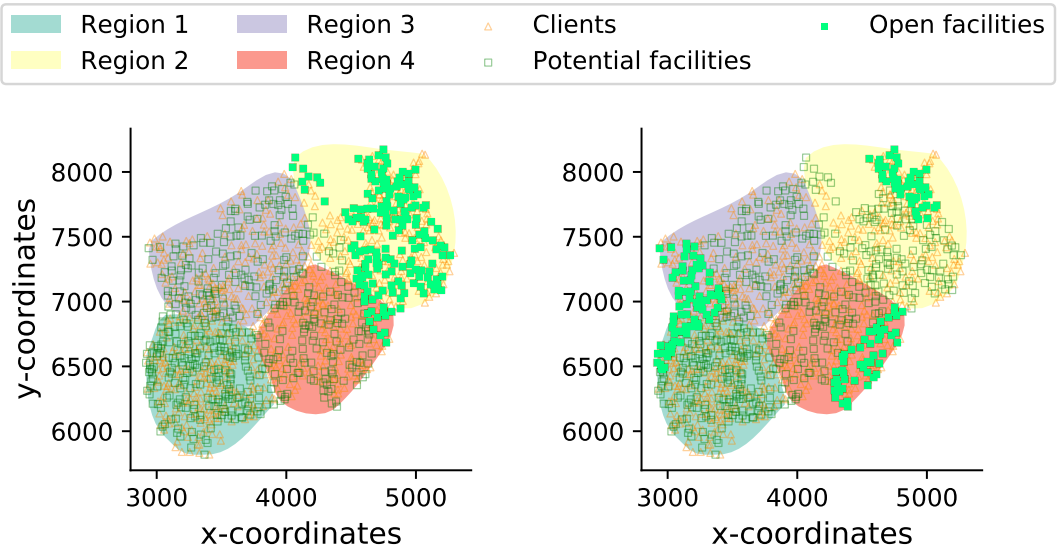

Figure 3.6.1: Solution derived by the matheuristic for the OPMP (left) and for the extension of the OPMP (right) for instance 59

3.7 Conclusion

In this paper, we consider the obnoxious p-median problem in which a set of clients and a set of potential locations for obnoxious facilities are given. From the set of potential locations, a given number of facilities must be opened. For each client, the distance to the closest open facility must be determined. The objective is to maximize the sum of these distances. Various extensions may arise in practice that include practical constraints on the locations of the obnoxious facilities. In this paper, we consider the classical obnoxious p-median problem and, in addition, an extension that includes upper bounds on the number of open facilities for different regions of an instance. In the literature, several exact and metaheuristic approaches have been introduced for the obnoxious p-median problem. However, while the exact approaches are not scalable to instances involving a large number of clients and potential locations, the metaheuristic approaches are not easily adaptable to extensions of the obnoxious p-median problem. We close this gap by introducing a novel matheuristic that is scalable to large instances and easily extendable to include practical constraints. Our matheuristic ensures that a large portion of the solution space is searched by starting from diverse initial solutions. It iteratively improves the initial solutions by exploring promising neighborhoods. In each iteration, a given number of facilities is removed and added by applying two adapted, appropriate versions of a model from the literature. We show that our matheuristic outperforms the leading metaheuristics from

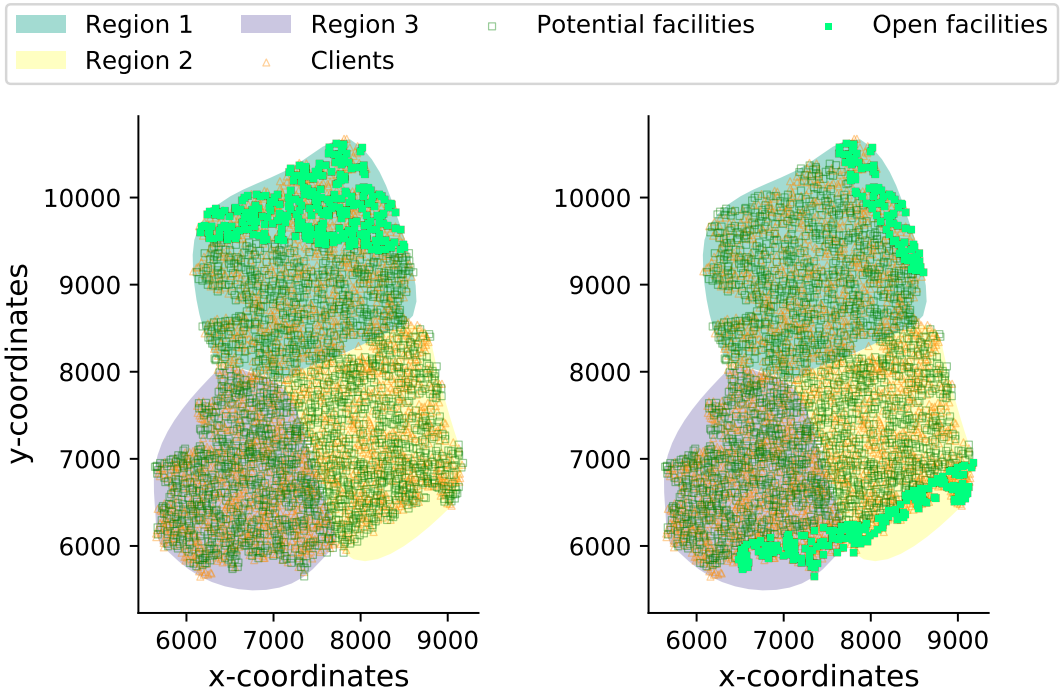

Figure 3.6.2: Solution derived by the matheuristic for the OPMP (left) and for the extension of the OPMP (right) for instance 61

the literature on large instances, including up to 21,000 clients and potential locations, and is competitive with them on small and medium instances. Furthermore, we illustrate the flexibility of our matheuristic by applying it to an extension of the obnoxious p-median problem with only minor changes made to the procedure.

A promising future research direction could be applying the matheuristic to other relevant extensions of the obnoxious p-median problem. For example, from a practical perspective, it could be interesting to consider upper bounds on the number of open facilities located within a certain distance of each client. These constraints would ensure that clients are not close to too many obnoxious facilities at once. Furthermore, the matheuristic could be adapted to other versions of the p-median problem. The general strategy of removing facilities from a solution and adding facilities to a partial solution using a model from the literature could also be applied to, for example, fault-tolerant versions where each client must be assigned to more than one open facility. As shown in this paper, our matheuristic could also easily be adapted to extensions of the p-median problem that include additional constraints such as capacities or fairness constraints.

Appendix

3.A Appendix A

Here, we describe how initial solutions are generated for the instances where only the distances d_{ij} between the clients and facilities are available but not the coordinates. We select a random client i and open the p facilities nearest to this client because we assume that these facilities are located close to each other. When the second initial solution is generated, we want to ensure that different areas of the solution space are covered. Hence, we select the facility j that is furthest away from the randomly selected client i from the first initial solution. Then, we select the closest client i' to facility j. We derive the second initial solution by opening the p closest facilities to client i'. We repeat the procedure if more than two initial solutions must be generated. Furthermore, two initial solutions are generated according to two simple rules. In the first rule, all facilities are sorted in descending order of their minimum distance to all clients $i \in I$. The first p facilities, i.e., the p facilities with the largest minimum distance, are opened (cf. Belotti et al., 2007). In the second rule, the p facilities with the highest sum of distances to all clients $i \in I$ are opened. Please note that the initial solution that results from the second rule represents an optimal solution to an adapted version of the OPMP in which the sum of distances between all clients and all facilities must be maximized. Additionally, also for the instances for which the coordinates are not given, two initial solutions are generated randomly.

3.B Appendix B

Here, we describe how we derive the set of considered facilities J'. In preliminary testing, we observed that facilities in good solutions are often located close to each other and far away from a center of the instance. Based on this observation, we developed a distance-based priority rule in which potential facilities that are close to an already open facility $j \in J^{part}$ and far away from a center of the instance have a high probability of being selected into J'. In detail, the distance-based priority rule works as follows.

First, we initialize $J' = \emptyset$. Then, we add all potential facilities to J' that were removed from the solution in step 2. By adding these potential facilities to J', we ensure that the solution resulting from the current iteration will have at least the same solution quality as the solution derived in the last iteration (given that the subproblems are solved to optimality). It is also important to ensure that there are additional potential facilities

in J' allowing the matheuristic to find improvements. We want to ensure that each of the b potential facilities that were removed from the solution in step 2 can potentially be replaced by another facility. Therefore, we compute the number of potential facilities that are additionally added to J' based on m^{max}, the number of potential facilities that can be considered in the model such that it can still be solved relatively quickly by a generic MILP solver, and b, the number of facilities that were removed from the solution in step 2. More precisely, we set the number of potential facilities that are added to J' in addition to the already added b potential facilities to $\max(m^{max} - b, b)$. In the following two paragraphs, for the sake of simplicity, we will refer to the potential facilities as facilities.

Next, we derive a set of facilities J^{near} from which we select the facilities into J'. For each facility $j \in J^{part}$, we sort the facilities in $J \setminus J^{sol}$ ascending in their distance to facility j. The set J^{near} then corresponds to the union of the m^{max} nearest facilities to each facility $j \in J^{part}$. Next, we compute a probability for each facility $j \in J^{near}$ of being selected into J'. These probabilities are computed as follows: we first compute the centroid of the facilities by averaging over each of the coordinates of the facilities $j \in J$. Then, we compute the Euclidean distance of each facility $j \in J^{near}$ to this centroid. We normalize the Euclidean distances by deducting their minimum and adding a value of 0.01 (to avoid having distances of exactly zero). Next, we divide them by their sum, which results in a probability value for each facility $j \in J^{near}$ of being selected into J'. Based on the derived probability values, we select $\max(m^{max} - b, b)$ facilities from J^{near} into J'. This distance-based priority rule has been shown to work substantially better than a random selection of facilities from J and better than a deterministic greedy selection of the facilities where the nearest facilities to the currently opened facilities are selected (cf. Section 3.5.5). In case $|J^{near}| \leq \max(m^{max} - b, b)$, all facilities from J^{near} are added to J'.

Note that for some instances, the coordinates v_i, $i \in I$, of the clients and w_j, $j \in J$, of the facilities are unavailable (cf. last paragraph of Section 3.4.2). Hence, the described procedure to derive J' cannot be applied. For these instances, the facilities can be randomly selected into J'. However, note that for the instances from the literature for which the coordinates of the clients and facilities are not given, $m^{max} \geq m$ in the experimental design used in Section 3.5.4. Hence, no selection criterion is needed, and J' is set to J in each iteration.

3.C Appendix C

Figures 3.C.1–3.C.5 illustrate Iterations 3–12 of the illustrative example when the matheuristic is applied. In Iterations 7–12, b is continuously increased because no improvement is found. The matheuristic stops when either the time limit is reached or $b = p$, i.e., all facilities are removed and added again, as is the case in Iteration 12.

3.D Appendix D

Table 3.D.1 shows information for the new instances.

Table 3.D.1: 69 small, medium, and large test instances for the OPMP

ID	Name	Size	n	m	p (level)	Shape	Source
1	xit1083	small	540	540	67 (low)	square	Rohe (2013)
2	xit1083	small	540	540	135 (medium)	square	Rohe (2013)
3	xit1083	small	540	540	270 (high)	square	Rohe (2013)
4	dka1376	small	688	688	86 (low)	square	Rohe (2013)
5	dka1376	small	688	688	172 (medium)	square	Rohe (2013)
6	dka1376	small	688	688	344 (high)	square	Rohe (2013)
7	icw1483	small	740	740	92 (low)	square	Rohe (2013)
8	icw1483	small	740	740	185 (medium)	square	Rohe (2013)
9	icw1483	small	740	740	370 (high)	square	Rohe (2013)
10	**fra1488**	**small**	**744**	**744**	**93 (low)**	**square**	**Rohe (2013)**
11	fra1488	small	744	744	186 (medium)	square	Rohe (2013)
12	fra1488	small	744	744	372 (high)	square	Rohe (2013)
13	dkd1973	small	986	986	123 (low)	square	Rohe (2013)
14	dkd1973	small	986	986	246 (medium)	square	Rohe (2013)
15	dkd1973	small	986	986	493 (high)	square	Rohe (2013)
16	irw2802	medium	1,400	1,400	175 (low)	square	Rohe (2013)
17	irw2802	medium	1,400	1,400	350 (medium)	square	Rohe (2013)
18	irw2802	medium	1,400	1,400	700 (high)	square	Rohe (2013)
19	**bgb4355**	**medium**	**2,177**	**2,177**	**272 (low)**	**square**	**Rohe (2013)**
20	bgb4355	medium	2,177	2,177	544 (medium)	square	Rohe (2013)

Continued on next page

Table 3.D.1: 69 small, medium, and large test instances for the OPMP

ID	Name	Size	n	m	p (level)	Shape	Source
21	bgb4355	medium	2,177	2,177	1,088 (high)	square	Rohe (2013)
22	xsc6880	medium	3,440	3,440	430 (low)	square	Rohe (2013)
23	**xsc6880**	**medium**	**3,440**	**3,440**	**860 (medium)**	**square**	**Rohe (2013)**
24	xsc6880	medium	3,440	3,440	1,720 (high)	square	Rohe (2013)
25	ida8197	medium	4,098	4,098	512 (low)	square	Rohe (2013)
26	ida8197	medium	4,098	4,098	1,024 (medium)	square	Rohe (2013)
27	ida8197	medium	4,098	4,098	2,049 (high)	square	Rohe (2013)
28	xrb14233	medium	7,116	7,116	889 (low)	square	Rohe (2013)
29	xrb14233	medium	7,116	7,116	1,779 (medium)	square	Rohe (2013)
30	**xrb14233**	**medium**	**7,116**	**7,116**	**3,558 (high)**	**square**	**Rohe (2013)**
31	pjh17845	large	8,922	8,922	1,115 (low)	square	Rohe (2013)
32	pjh17845	large	8,922	8,922	2,230 (medium)	square	Rohe (2013)
33	**pjh17845**	**large**	**8,922**	**8,922**	**4,461 (high)**	**square**	**Rohe (2013)**
34	ido21215	large	10,607	10,607	1,325 (low)	square	Rohe (2013)
35	ido21215	large	10,607	10,607	2,651 (medium)	square	Rohe (2013)
36	ido21215	large	10,607	10,607	5,303 (high)	square	Rohe (2013)
37	dkd1973	small	1,479	493	61 (low)	square	Rohe (2013)
38	dkd1973	small	1,479	493	123 (medium)	square	Rohe (2013)
39	dkd1973	small	1,479	493	246 (high)	square	Rohe (2013)
40	ida8197	medium	6,147	2,049	256 (low)	square	Rohe (2013)
41	ida8197	medium	6,147	2,049	512 (medium)	square	Rohe (2013)
42	ida8197	medium	6,147	2,049	1,024 (high)	square	Rohe (2013)
43	ido21215	large	15,911	5,303	662 (low)	square	Rohe (2013)
44	ido21215	large	15,911	5,303	1,325 (medium)	square	Rohe (2013)
45	ido21215	large	15,911	5,303	2,651 (high)	square	Rohe (2013)
46	dkd1973	small	493	1,479	184 (low)	square	Rohe (2013)
47	dkd1973	small	493	1,479	369 (medium)	square	Rohe (2013)
48	dkd1973	small	493	1,479	739 (high)	square	Rohe (2013)
49	ida8197	medium	2,049	6,147	768 (low)	square	Rohe (2013)
50	ida8197	medium	2,049	6,147	1,536 (medium)	square	Rohe (2013)
51	ida8197	medium	2,049	6,147	3,073 (high)	square	Rohe (2013)

Continued on next page

Table 3.D.1: 69 small, medium, and large test instances for the OPMP

ID	Name	Size	n	m	p (level)	Shape	Source
52	ido21215	large	5,303	15,911	1,988 (low)	square	Rohe (2013)
53	ido21215	large	5,303	15,911	3,977 (medium)	square	Rohe (2013)
54	ido21215	large	5,303	15,911	7,955 (high)	square	Rohe (2013)
55	dsj1000	small	500	500	62 (low)	other	Reinelt (1991)
56	dsj1000	small	500	500	125 (medium)	other	Reinelt (1991)
57	dsj1000	small	500	500	250 (high)	other	Reinelt (1991)
58	nrw1379	small	689	689	86 (low)	other	Reinelt (1991)
59	nrw1379	small	689	689	172 (medium)	other	Reinelt (1991)
60	nrw1379	small	689	689	344 (high)	other	Reinelt (1991)
61	fnl4461	medium	2,230	2,230	278 (low)	other	Reinelt (1991)
62	**fnl4461**	**medium**	**2,230**	**2,230**	**557 (medium)**	**other**	**Reinelt (1991)**
63	fnl4461	medium	2,230	2,230	115 (high)	other	Reinelt (1991)
64	brd14051	medium	7,025	7,025	878 (low)	other	Reinelt (1991)
65	brd14051	medium	7,025	7,025	1,756 (medium)	other	Reinelt (1991)
66	**brd14051**	**medium**	**7,025**	**7,025**	**3,512 (high)**	**other**	**Reinelt (1991)**
67	d15112	medium	7,556	7,556	944 (low)	other	Reinelt (1991)
68	d15112	medium	7,556	7,556	1,889 (medium)	other	Reinelt (1991)
69	d15112	medium	7,556	7,556	3,778 (high)	other	Reinelt (1991)

3.E Appendix E

Table 3.E.1 shows the objective function values (OFVs) derived by the matheuristic, the PISTS metaheuristic, and the IG metaheuristic for the benchmark instances from the literature. Table 3.E.2 shows the objective function values (OFVs) derived by the three approaches for the new instances. The best values are highlighted in bold.

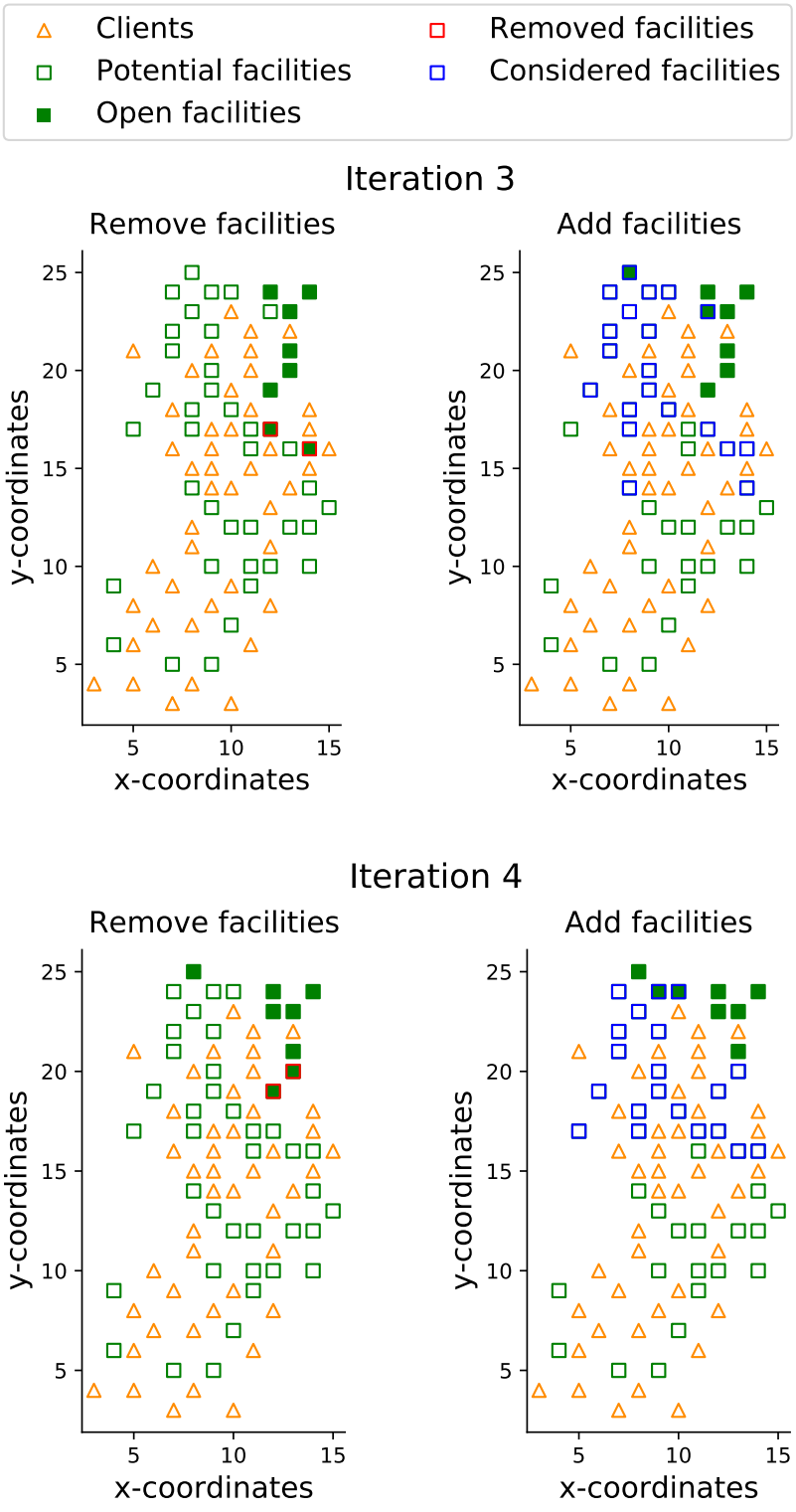

Figure 3.C.1: Iterations 3 and 4 of the matheuristic for the illustrative example

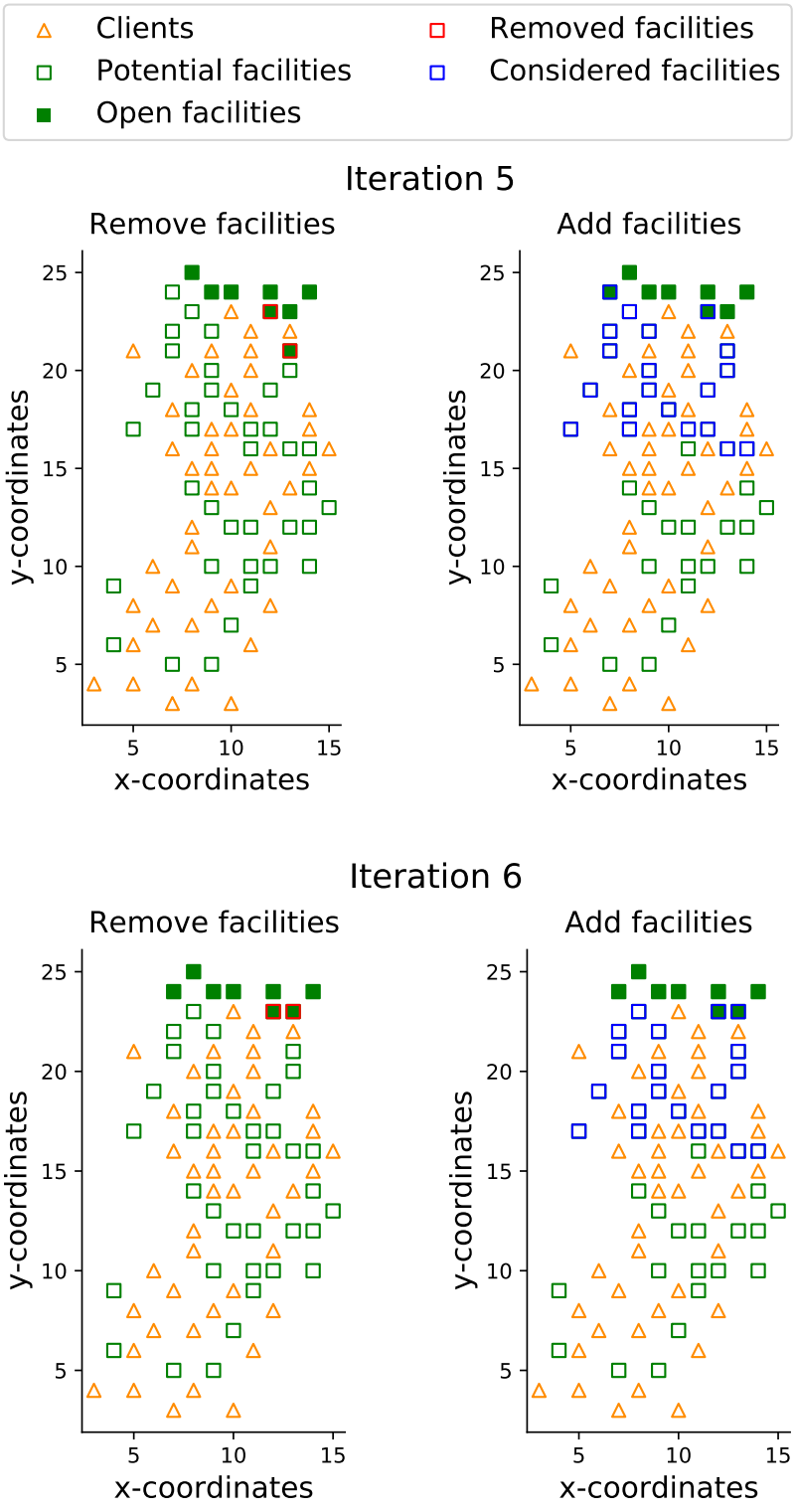

Figure 3.C.2: Iterations 5 and 6 of the matheuristic for the illustrative example

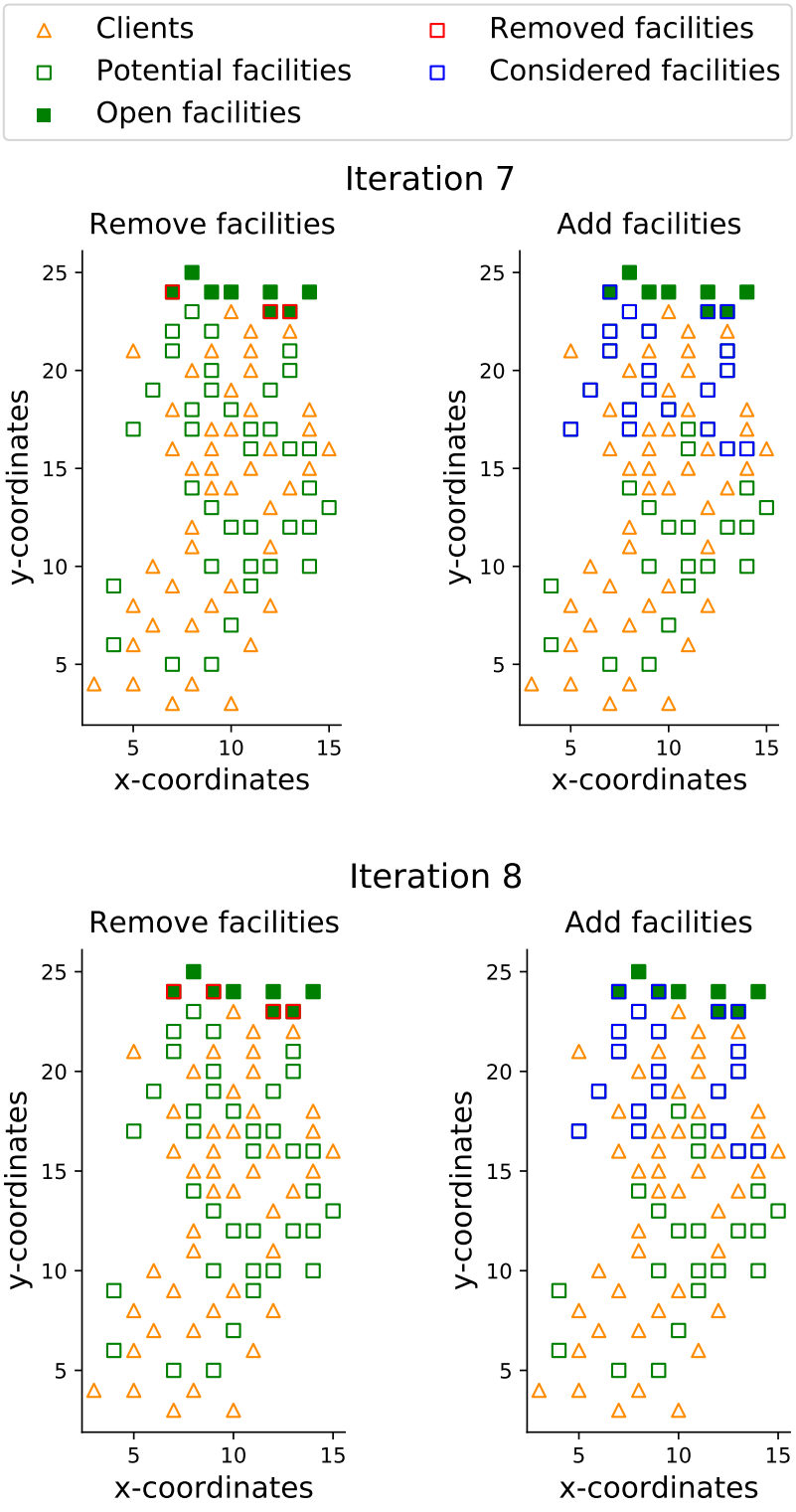

Figure 3.C.3: Iterations 7 and 8 of the matheuristic for the illustrative example

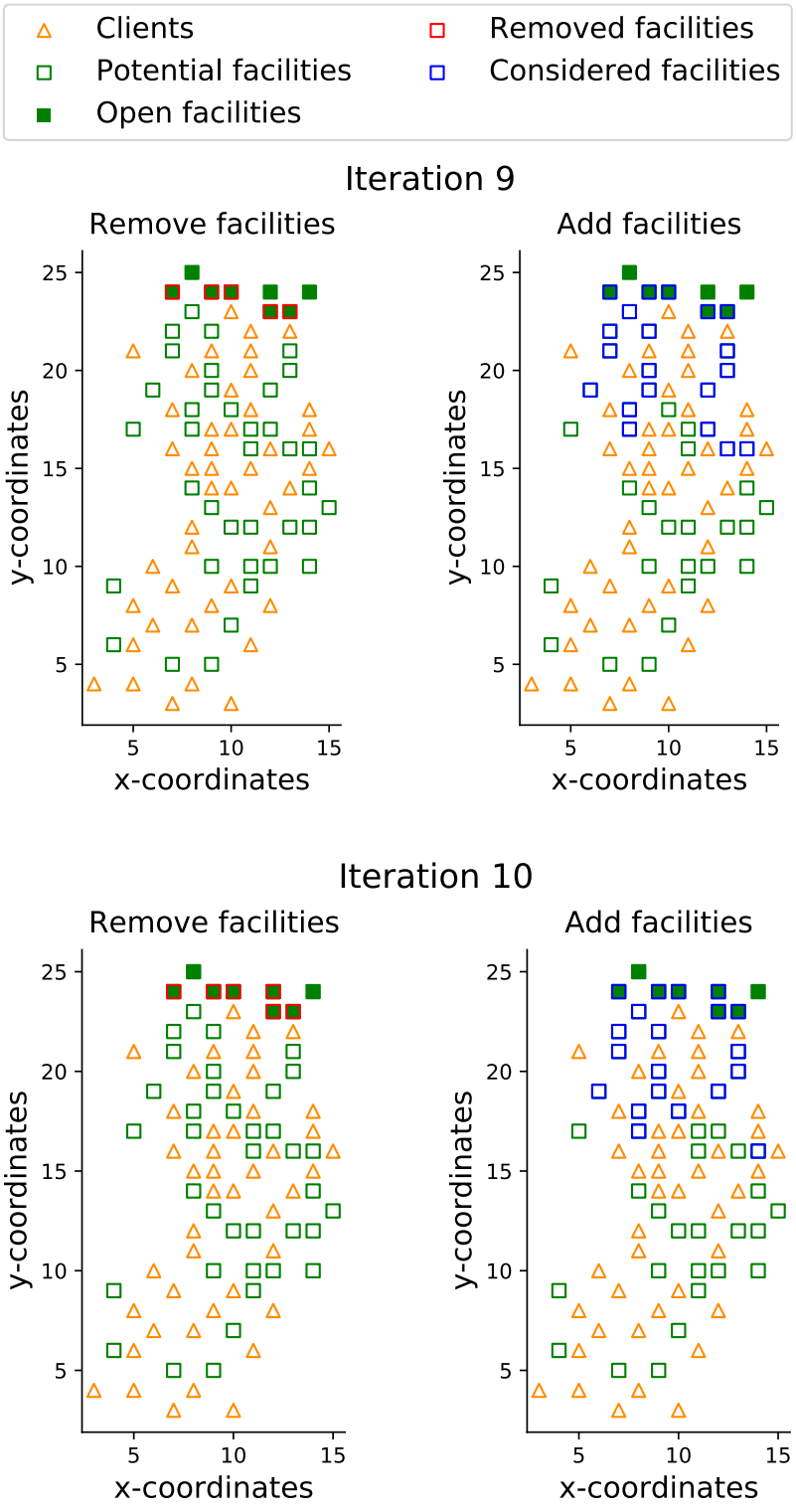

Figure 3.C.4: Iterations 9 and 10 of the matheuristic for the illustrative example

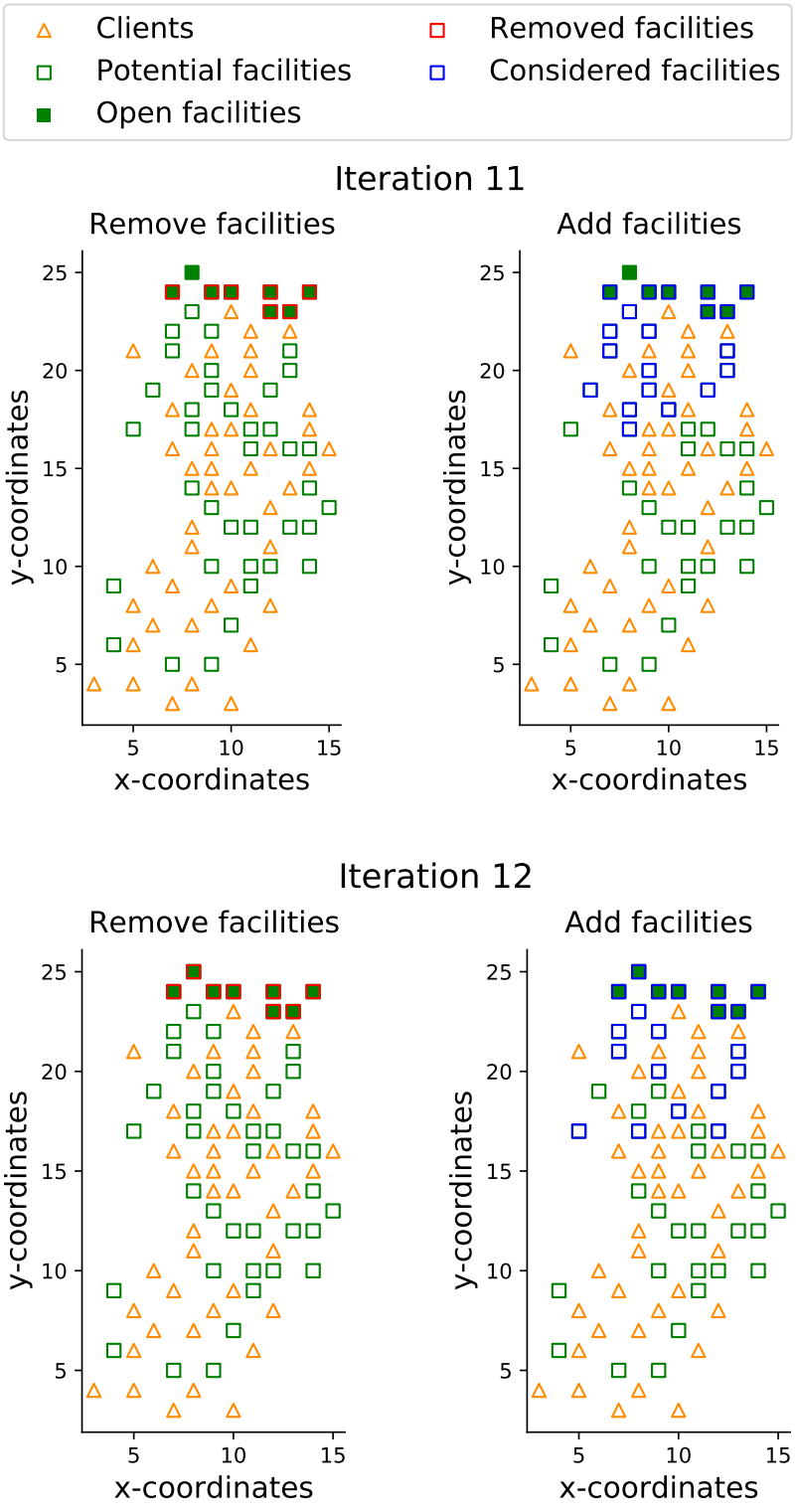

Figure 3.C.5: Iterations 11 and 12 of the matheuristic for the illustrative example

Table 3.E.1: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the benchmark instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
pmed17-p25.A	7,317	7,317	7,317
pmed17-p50.A	5,411	5,411	5,411
pmed17-p100.A	4,054	4,054	4,054
pmed18-p25.A	7,432	7,432	7,432
pmed18-p50.A	5,746	5,746	5,746
pmed18-p100.A	4,220	4,220	4,220
pmed19-p25.A	7,020	7,020	7,020
pmed19-p50.A	5,387	5,387	5,387
pmed19-p100.A	4,033	4,033	4,033
pmed20-p25.A	7,648	7,648	7,648
pmed20-p50.A	5,872	5,872	5,872
pmed20-p100.A	4,063	4,063	4,063
pmed21-p31.A	7,304	7,304	7,304
pmed21-p62.A	5,784	5,775	5,784
pmed21-p125.A	4,155	4,155	4,155
pmed22-p31.A	7,900	7,900	7,900
pmed22-p62.A	5,995	5,995	5,995
pmed22-p125.A	4,358	4,358	4,358
pmed23-p31.A	7,841	7,841	7,841
pmed23-p62.A	5,785	5,785	5,785
pmed23-p125.A	4,114	4,114	4,114
pmed24-p31.A	7,425	7,425	7,425
pmed24-p62.A	5,528	5,528	5,528
pmed24-p125.A	4,091	4,091	4,091
pmed25-p31.A	7,552	7,552	7,552
pmed25-p62.A	5,767	5,767	5,767
pmed25-p125.A	4,155	4,155	4,155
pmed26-p37.A	8,112	8,112	8,112
pmed26-p75.A	5,789	5,789	5,789
pmed26-p150.A	4,339	4,339	**4,341**

Continued on next page

Table 3.E.1: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the benchmark instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
pmed27-p37.A	7,556	7,556	7,556
pmed27-p75.A	5,668	5,668	5,668
pmed27-p150.A	4,062	4,052	4,062
pmed28-p37.A	7,366	7,366	7,366
pmed28-p75.A	5,681	5,681	5,681
pmed28-p150.A	4,099	4,099	4,099
pmed29-p37.A	7,395	7,404	7,404
pmed29-p75.A	5,880	5,880	5,880
pmed29-p150.A	4,141	4,140	4,141
pmed30-p37.A	7,704	7,704	7,704
pmed30-p75.A	6,184	6,184	**6,189**
pmed30-p150.A	4,385	4,385	4,385
pmed31-p43.A	7,424	7,424	7,424
pmed31-p87.A	5,905	5,905	5,905
pmed31-p175.A	4,136	4,135	4,136
pmed32-p43.A	7,794	7,794	7,794
pmed32-p87.A	5,925	5,925	5,925
pmed32-p175.A	4,242	4,242	4,242
pmed33-p43.A	7,598	7,598	7,598
pmed33-p87.A	5,793	5,793	5,793
pmed33-p175.A	4,105	4,103	4,105
pmed34-p43.A	7,725	7,725	7,725
pmed34-p87.A	5,844	5,849	5,849
pmed34-p175.A	4,287	4,286	4,287
pmed35-p50.A	7,155	7,155	7,155
pmed35-p100.A	5,845	5,845	5,845
pmed35-p200.A	4,007	4,005	4,007
pmed36-p50.A	8,179	8,179	8,179
pmed36-p100.A	6,461	6,461	6,461
pmed36-p200.A	4,318	4,317	**4,319**

Continued on next page

Table 3.E.1: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the benchmark instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
pmed37-p50.A	7,830	7,830	7,830
pmed37-p100.A	6,203	6,203	6,203
pmed37-p200.A	4,593	4,588	4,593
pmed38-p56.A	7,432	7,432	7,432
pmed38-p112.A	5,912	5,914	**5,915**
pmed38-p225.A	4,428	4,428	4,428
pmed39-p56.A	7,712	7,712	7,712
pmed39-p112.A	5,935	5,935	5,935
pmed39-p225.A	4,369	4,368	4,369
pmed40-p56.A	8,211	8,211	8,211
pmed40-p112.A	6,272	6,272	6,272
pmed40-p225.A	4,572	4,572	4,572
pmed17-p25.B	6,905	6,905	6,905
pmed17-p50.B	5,563	5,563	5,563
pmed17-p100.B	3,992	3,992	3,992
pmed18-p25.B	7,662	7,662	7,662
pmed18-p50.B	5,852	5,852	5,852
pmed18-p100.B	4,122	4,122	4,122
pmed19-p25.B	6,816	6,816	6,816
pmed19-p50.B	5,423	5,423	5,423
pmed19-p100.B	4,016	4,016	4,016
pmed20-p25.B	7,349	7,349	7,349
pmed20-p50.B	5,665	5,665	5,665
pmed20-p100.B	4,067	4,067	4,067
pmed21-p31.B	7,331	7,331	7,331
pmed21-p62.B	5,870	5,870	5,870
pmed21-p125.B	4,033	4,033	4,033
pmed22-p31.B	7,695	7,695	7,695
pmed22-p62.B	6,259	6,259	6,259
pmed22-p125.B	4,338	4,338	4,338

Continued on next page

Table 3.E.1: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the benchmark instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
pmed23-p31.B	7,137	7,137	7,137
pmed23-p62.B	5,724	5,724	5,724
pmed23-p125.B	4,095	4,095	4,095
pmed24-p31.B	7,190	7,190	7,190
pmed24-p62.B	5,752	5,752	5,752
pmed24-p125.B	4,072	4,072	4,072
pmed25-p31.B	7,552	7,552	7,552
pmed25-p62.B	5,692	5,692	5,692
pmed25-p125.B	4,233	4,227	4,233
pmed26-p37.B	7,643	7,643	7,643
pmed26-p75.B	5,923	5,923	5,923
pmed26-p150.B	4,173	4,173	4,173
pmed27-p37.B	7,448	7,448	7,448
pmed27-p75.B	5,844	5,844	5,844
pmed27-p150.B	4,144	4,144	4,144
pmed28-p37.B	7,388	7,388	7,388
pmed28-p75.B	5,642	5,633	5,642
pmed28-p150.B	4,069	4,069	4,069
pmed29-p37.B	7,529	7,529	7,529
pmed29-p75.B	5,709	5,709	5,709
pmed29-p150.B	4,157	4,157	4,157
pmed30-p37.B	8,048	8,048	8,048
pmed30-p75.B	6,040	6,041	6,041
pmed30-p150.B	4,313	4,313	4,313
pmed31-p43.B	7,320	7,320	7,320
pmed31-p87.B	5,620	5,621	5,621
pmed31-p175.B	4,138	4,138	4,138
pmed32-p43.B	7,899	7,899	7,899
pmed32-p87.B	5,852	5,836	5,852
pmed32-p175.B	4,244	4,244	4,244

Continued on next page

Table 3.E.1: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the benchmark instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
pmed33-p43.B	7,611	7,611	7,611
pmed33-p87.B	5,840	5,840	5,840
pmed33-p175.B	4,156	4,153	4,156
pmed34-p43.B	7,514	7,514	7,514
pmed34-p87.B	5,857	5,857	5,857
pmed34-p175.B	4,270	4,270	4,270
pmed35-p50.B	7,570	7,570	7,570
pmed35-p100.B	5,639	5,639	5,639
pmed35-p200.B	4,109	4,109	4,109
pmed36-p50.B	8,144	8,144	8,144
pmed36-p100.B	6,219	6,200	6,219
pmed36-p200.B	4,321	4,319	4,321
pmed37-p50.B	8,379	8,379	8,379
pmed37-p100.B	6,210	6,203	**6,212**
pmed37-p200.B	4,609	4,609	4,609
pmed38-p56.B	7,532	7,535	7,535
pmed38-p112.B	5,949	5,949	5,949
pmed38-p225.B	4,446	4,445	4,446
pmed39-p56.B	7,625	7,625	7,625
pmed39-p112.B	6,198	6,198	6,198
pmed39-p225.B	**4,268**	4,265	4,267
pmed40-p56.B	8,022	8,022	8,022
pmed40-p112.B	6,200	6,199	6,200
pmed40-p225.B	4,532	4,532	4,532

Table 3.E.2: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the new instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
1	55,980	56,159	56,159
2	42,330	42,330	42,330
3	19,501	19,501	19,501
4	65,882	66,073	66,073
5	45,204	45,204	44,250
6	23,205	23,205	23,205
7	76,073	76,073	76,073
8	59,786	59,786	59,786
9	27,442	27,442	27,442
10	98,673	98,739	98,739
11	71,988	71,988	71,988
12	33,370	32,159	33,370
13	229,564	229,564	229,564
14	183,637	183,637	183,637
15	**75,969**	75,063	75,063
16	221,718	221,718	221,718
17	162,768	162,768	162,768
18	79,340	30,731	79,340
19	**303,304**	186,079	302,835
20	**225,823**	35,895	225,160
21	107,396	13,736	**108,286**
22	**739,146**	65,895	738,217
23	533,201	31,517	533,201
24	**246,009**	18,285	244,955
25	915,182	51,995	**915,691**
26	684,514	27,889	**689,318**
27	329,720	17,615	**330,311**
28	2,327,042	81,179	**2,367,170**
29	**1,805,546**	53,124	1,802,959
30	**833,905**	35,208	628,184

Continued on next page

Table 3.E.2: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the new instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
31	**3,035,021**	78,543	2,005,425
32	**2,283,958**	53,472	1,105,949
33	**1,042,475**	36,524	529,401
34	**2,925,845**	104,218	824,377
35	**1,882,518**	70,590	541,078
36	**904,664**	47,903	304,831
37	344,665	344,665	344,665
38	278,501	278,501	278,501
39	**121,824**	110,982	110,982
40	1,342,871	395,335	**1,400,903**
41	1,079,035	97,887	**1,079,836**
42	**466,468**	44,005	444,802
43	**4,339,605**	233,102	4,231,777
44	**2,891,112**	153,969	2,249,735
45	1,383,672	107,086	**1,554,307**
46	118,038	110,298	118,038
47	88,181	88,181	88,181
48	40,164	40,164	34,749
49	453,398	17,981	**453,708**
50	342,088	10,973	**342,136**
51	**162,451**	6,948	162,164
52	**1,439,845**	41,330	445,221
53	**967,498**	28,253	281,518
54	**408,555**	19,063	139,627
55	282,292,767	282,872,870	282,872,870
56	185,590,004	185,628,514	185,628,514
57	106,074,079	106,074,079	106,074,079
58	765,743	765,743	765,743
59	506,857	469,701	506,857

Continued on next page

Table 3.E.2: Objective function values derived by the matheuristic, PISTS metaheuristic, and IG metaheuristic for the new instances (note that higher values are better)

ID	OFV matheuristic	OFV PISTS	OFV IG
60	246,398	246,398	246,398
61	4,023,975	2,789,450	4,023,975
62	2,855,926	555,675	2,855,926
63	1,439,070	173,932	1,439,070
64	22,162,973	652,892	**22,209,219**
65	13,809,139	445,238	**14,414,789**
66	7,252,232	320,394	**7,651,457**
67	55,179,625	2,204,329	**57,293,470**
68	43,018,826	1,491,705	**43,556,600**
69	**21,623,304**	1,082,564	20,292,880

Bibliography

Belotti, P., Labbé, M., Maffioli, F., Ndiaye, M.M., 2007. A branch-and-cut method for the obnoxious p-median problem. 4OR 5, 299–314.

Chang, J., Wang, L., Hao, J.K., Wang, Y., 2021. Parallel iterative solution-based tabu search for the obnoxious p-median problem. Computers & Operations Research 127, 105155.

Chiang, Y.I., Lin, C.C., 2017. Compact model for the obnoxious p-median problem. American Journal of Operations Research 7, 348–355.

Church, R.L., Drezner, Z., 2022. Review of obnoxious facilities location problems. Computers & Operations Research 138, 105468.

Colmenar, J.M., Greistorfer, P., Martí, R., Duarte, A., 2016. Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem. European Journal of Operational Research 252, 432–442.

Colmenar, J.M., Martí, R., Duarte, A., 2018. Multi-objective memetic optimization for the bi-objective obnoxious p-median problem. Knowledge-Based Systems 144, 88–101.

Gokalp, O., 2020. An iterated greedy algorithm for the obnoxious p-median problem. Engineering Applications of Artificial Intelligence 92, 103674.

Herrán, A., Colmenar, J.M., Martí, R., Duarte, A., 2020. A parallel variable neighborhood search approach for the obnoxious p-median problem. International Transactions in Operational Research 27, 336–360.

Kalczynski, P., Drezner, Z., 2021. The obnoxious facilities planar p-median problem. OR Spectrum 43, 577–593.

Kalczynski, P., Drezner, Z., 2022. The obnoxious facilities planar p-median problem with variable sizes. Omega 111, 102639.

Kalczynski, P., Suzuki, A., Drezner, Z., 2020. Obnoxious facility location: The case of weighted demand points. ArXiv preprint arXiv:2008.04386.

Kalczynski, P., Suzuki, A., Drezner, Z., 2022. Obnoxious facility location in multiple dimensional space. TOP, 1–24.

Koch, T., Berthold, T., Pedersen, J., Vanaret, C., 2022. Progress in mathematical programming solvers from 2001 to 2020. EURO Journal on Computational Optimization 10, 100031.

Labbé, M., Maffioli, F., Ndiaye, M.M., Belotti, P., 2001. Obnoxious p-median problems: valid inequalities and a branch-and-cut approach. The OR Peripatetic Post-Graduate Programme, 26–29.

Lei, T.L., Church, R.L., 2015. On the unified dispersion problem: Efficient formulations and exact algorithms. European Journal of Operational Research 241, 622–630.

Lin, C.C., Chiang, Y.I., 2021. Alternative formulations for the obnoxious p-median problem. Discrete Applied Mathematics 289, 366–373.

Lin, G., Guan, J., 2018. A hybrid binary particle swarm optimization for the obnoxious p-median problem. Information Sciences 425, 1–17.

Maniezzo, V., Boschetti, M.A., Stützle, T., 2021. Preface, in: Maniezzo, V., Boschetti, M.A., Stützle, T. (Eds.), Matheuristics. Springer, Cham, pp. 143–158.

Mladenović, N., Alkandari, A., Pei, J., Todosijević, R., Pardalos, P.M., 2020. Less is more approach: basic variable neighborhood search for the obnoxious p-median problem. International Transactions in Operational Research 27, 480–493.

Reinelt, G., 1991. TSPLIB—A traveling salesman problem library. ORSA Journal on Computing 3, 376–384.

ReVelle, C.S., Eiselt, H.A., 2005. Location analysis: A synthesis and survey. European Journal of Operational Research 165, 1–19.

Rohe, A., 2013. VLSI collection. Website. `http://www.math.uwaterloo.ca/tsp/vlsi/index.html`. Accessed: 2022-06-20.

Sánchez-Oro, J., López-Sánchez, A.D., Colmenar, J.M., 2022. A multi-objective parallel variable neighborhood search for the bi-objective obnoxious p-median problem. Optimization Letters 16, 301–331.

Tamir, A., 1991. Obnoxious facility location on graphs. SIAM Journal on Discrete Mathematics 4, 550–567.