Applications of Mathematical Programming in Personnel Scheduling

INAUGURALDISSERTATION

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

vorgelegt von

Tom Rihm

Bern, April 2017

Die Fakultät hat diese Arbeit am 18. Mai 2017 auf Antrag der beiden Gutachter Prof. Dr. Norbert Trautmann und Prof. Dr. Rubén Ruiz als Dissertation angenommen, ohne damit zu den darin ausgesprochenen Auffassungen Stellung nehmen zu wollen.

Abstract

In the few decades of its existence, mathematical programming has evolved into an important branch of operations research and management science. This thesis consists of four papers in which we apply mathematical programming to real-life personnel scheduling and project management problems. We develop exact mathematical programming formulations. Furthermore, we propose effective heuristic strategies to decompose the original problems into subproblems that can be solved efficiently with tailored mathematical programming formulations. We opt for solution methods that are based on mathematical programming, because their advantages in practice are a) the flexibility to easily accommodate changes in the problem setting, b) the possibility to evaluate the quality of the solutions obtained, and c) the possibility to use general-purpose solvers, which are often the only software available in practice.

The present dissertation includes the following four papers.

Paper I

Rihm, T., Trautmann, N. (2017). A decomposition heuristic for short-term planning of assessment centres. *European Journal of Industrial Engineering 11(6), 725–751.* DOI: 10.1504/EJIE.2017.10006711.

Paper II

Rihm, T., Trautmann, N., Zimmermann, A. (2016). MIP formulations for an application of project scheduling in human resource management. *Flexible Services and Manufacturing Journal.* Advance online publication. DOI: 10.1007/s10696-016-9260-8.

Paper III

Rihm, T. (2017). Minimizing operational costs of assessment centers. *In: Kaihara, T., Nonobe, K. (eds.). Proceedings of the International Symposium on Scheduling 2017. Nagoya, 45–50.*

Paper IV

Rihm, T., Baumann, P. (2018). Staff assignment with lexicographically ordered acceptance levels. *Journal of Scheduling 21(2), 167-189.* DOI: 10.1007/s10951-017-0525-1.

Acknowledgements

I elaborated this thesis during my time as a PhD student at the Department of Business Administration at the University of Bern. I would like to thank all those people who have supported and encouraged me and made this time a great experience for me.

First and foremost, I would like to thank my supervisor Prof. Dr. Norbert Trautmann for offering a stimulating working environment, continuous support, and valuable advices throughout my PhD studies. Moreover, I am grateful to Prof. Dr. Philipp Baumann and Dr. Adrian Zimmermann for good cooperation on joint projects.

Special thanks go to all my colleagues from the Chair of Quantitative Methods for bouncing ideas back and forth. You have contributed immensely to my personal and professional time in Bern.

Moreover, I would like to thank Prof. Dr. Rubén Ruiz for his willingness to act as an external referee of my thesis.

Finally, I would like to express my gratitude to my parents, Josette Wampach and Serge Rihm, and my sister Michèle Rihm who have been a great support during my whole life.

Bern, April 2017

Tom Rihm

Contents

Introduction

Personnel scheduling is an essential and recurring challenge for many companies and organizations, especially in the service industries. In general, personnel scheduling involves a) the determination of the tasks (or shifts) to be performed, b) the temporal scheduling of these tasks, and c) the assignment of the employees to these tasks. As manpower is the most critical and expensive resource for many companies, a careful and proper planning leads to substantial cost savings for the company and a greater job satisfaction for the employees. However, generating a personnel schedule is a very challenging and time-consuming process, as a large variety of different and often conflicting requests have to be considered simultaneously.

Mathematical programming-based planning tools can significantly speed up the schedule-generation process and considerably improve the resulting schedules. In the few decades of its existence, mathematical programming has evolved into an important branch of operations research and management science (cf., e.g., Bixby and Rothberg, 2007; Williams, 2013). Mathematical programming is concerned with determining an optimal solution to a planning problem and assists this way in taking decisions. It encompasses modeling techniques to formulate planning problems as mathematical models and algorithms to solve them. A typical application of mathematical programming is allocating some scarce resources, e.g., employees or machines, such that a given objective is maximized, e.g., profit or employee satisfaction. For industrial practitioners, the advantages of mathematical programming are a) the flexibility to easily accommodate changes in the problem setting, b) the possibility to evaluate the quality of the solutions obtained, and c) the possibility to use general-purpose solvers, which are often the only software available to industrial practitioners (cf., e.g., Koné et al., 2011; Kopanos et al., 2014).

The formulation of a planning problem as a mathematical model is not unique. In general, different formulations can be used to model the same planning problem. Because the performance of general-purpose solvers strongly depends on the underlying formulation (cf., e.g., Vielma, 2015), analyzing different formulations for each planning problem is vitally important. A drawback of mathematical programming-based models is that

they contain a substantial number of constraints and variables for large-sized instances. Therefore, despite the recent improvements in optimization software and computer hardware (cf., e.g., Lodi, 2010; Koch et al., 2011; Bixby, 2012), often no feasible solution is found in a reasonable amount of computation time for large-sized instances.

This thesis consists of four papers on personnel scheduling. For various specific real-life scheduling problems, we develop novel and efficient mathematical programming-based solution approaches. Hence, they can be adapted easily to changes in the problem setting or to related scheduling problems. For the treatment of large-sized instances, we propose effective heuristic strategies to decompose the original problem into subproblems that can be solved efficiently with tailored mathematical programming formulations. These strategies have two important advantages: a) we maintain the flexibility of mathematical programming to easily accommodate complex constraints, and b) we are able to control the speed of the optimization behavior by the size of the subproblems. In practice, both aspects are particularly important, i.e., when short computation time limits are prescribed or when the problem settings change dynamically.

In the first paper, we consider the assessment center planning problem (ACP). The ACP originates from a human resource management service provider that conducts assessment centers for corporate clients, e.g., banks. In an assessment center, candidates for job positions perform different tasks while being observed and evaluated by so-called assessors. The planning problem consists of scheduling all tasks and a lunch break for each candidate and determining which assessors are assigned to which candidate during which task. Because the assessors are usually senior managers of the company or highly qualified psychologists, the objective of the ACP is to minimize the total waiting time for the assessors. Specific rules for assigning the assessors to the candidates distinguish the ACP from related scheduling problems discussed in the literature (e.g., the resource-constrained project scheduling problem). In particular, because of fairness considerations, the number of different assessors assigned to a candidate at least once must be approximately the same for each candidate. Due to these application-specific restrictions, we cannot apply solution methods from the literature straightforwardly to the ACP. We present a decomposition heuristic that separates the scheduling and assignment decisions into different subproblems. We then solve each subproblem using an appropriate mixed-integer linear programming (MIP) formulation. The scheduling decisions determine the start times for the tasks, whereas the assignment decisions assign the assessors to the tasks. In general, heuristics are unable to evaluate the quality of the solution found. However, a salient feature of the proposed decomposition heuristic is that the scheduling subproblem provides a strong lower bound for the original problem. In a computational analysis, we apply

this decomposition heuristic to 4 real-life instances and to 240 systematically generated test instances derived from real-life data. Our computational results demonstrate that this novel heuristic is able, for the first time, to solve the four real-life instances to optimality. Furthermore, this heuristic outperforms the state-of-the-art approaches, i.e., the list-scheduling heuristic of Zimmermann and Trautmann (2015) and the MIP formulations of Grüter et al. (2014), Zimmermann and Trautmann (2014), and Rihm et al. (2016).

In the second paper, we deal with the same assessment center planning problem as in the first paper. We develop problem-specific lower bounds and analyze different MIP formulations for the ACP. In detail, we provide two discrete-time and three continuous-time MIP formulations. In discrete-time formulations, the planning horizon is divided into a set of time intervals of equal length, and the activities can only start at the endpoints of these intervals. Conversely, in continuous-time formulations, the activities can start at any point in time. In a comparative study, we analyze the strength of the lower bounds and the performance of the five MIP formulations for the same instances as in the first paper. The results demonstrate that for all instances, the developed lower bounds are very close or equal to the optimal objective function values. Furthermore, the MIP formulations provide good or optimal solutions within reasonable computational time. Surprisingly, in contrast to results generally obtained for related planning problems (e.g., the resource constrained project scheduling problem), the continuous-time formulations outperform the discrete-time formulations in solution quality. However, we obtain the best MIP-based lower bounds using the discrete-time formulations.

In the third paper, we study a novel planning problem in the context of assessment centers, which we call the assessment center resource investment problem (ACRIP). In the ACRIP, the goal is to minimize the total operational costs to meet a given deadline and the constraints from the ACP. The operational costs increase with additional assessors, actors, or rooms. In contrast to the ACP that we study in the first and the second paper, the number of required assessors and actors is to be determined. The literature on project scheduling makes a similar distinction between the objective functions. Minimizing the total duration for given resource capacities is referred to as the resource-constrained project scheduling problem (cf., e.g., Artigues et al., 2015), and minimizing the total resource costs as per a project completion deadline is referred to as the resource investment problem (cf., e.g., Möhring, 1984). However, owing to the problem-specific rules for assigning the assessors to the tasks and the candidates, the solution methods for the resource investment problem are not applicable to the ACRIP. Hence, we develop a novel discrete-time MIP formulation to solve the ACRIP. We choose a discrete-time formulation because in the second paper it turned out that discrete-time formulations

yield the best lower bounds for a related planning problem. To speed up the search process, we propose some preprocessing techniques and a novel row generation scheme that exploits the structural properties of the ACRIP. In this scheme, some constraints that mainly drive the computation time of a general-purpose solver are relaxed and the relaxed formulation is solved. Whenever an integer solution is found that violates one of the relaxed constraints, a violated constraint is added to the formulation and a heuristic attempts to transform the integer solution into a feasible integer solution. In a computational study, we test the MIP formulation with and without the row generation scheme on a set of instances derived from real-life data. The results highlight a great potential to save operational costs of assessment centers. Furthermore, using the row generation scheme increases the performance of the general-purpose solver considerably.

In the fourth paper, we study a new real-life staff assignment problem, the staff assignment problem with lexicographically ordered acceptance levels (SAP-LAL). The SAP-LAL consists of assigning employees to work shifts subject to a large variety of critical and non-critical requests, including personal preferences of employees. This problem was reported to us by a provider of commercial employee scheduling software that has developed a new user interface to specify trade-offs among different requests. The user defines a target value for each request and assigns integer acceptance levels to deviations from this target value. These acceptance levels reflect the relative severity of possible deviations, e.g., for an employee that requests at least two weekends off, obtaining one weekend off is preferable to no weekend off, and thus receives a higher acceptance level. The objective is to minimize the total number of deviations in lexicographical order of the acceptance levels. This objective cannot be represented straightforwardly in existing staff assignment approaches from the literature, because each request is associated with several acceptance levels. We provide a binary linear programming formulation, propose novel aggregation techniques to reduce the size of the formulation, and develop a MIP-based heuristic for large-sized instances. The main methodological feature of the MIP-based heuristic is an employee selection rule for effectively decomposing the original problem into subproblems. In a computational analysis, we apply the binary linear programming formulation and the MIP-based heuristic to a real-world instance and a test set that contains 45 instances derived from real-life data. Our computational analysis shows that the two approaches solve small- and medium-sized instances to optimality. Furthermore, the MIP-based heuristic delivers high-quality solutions for large-sized instances with limited computational effort and outperforms the commercial employee scheduling software of our industry partner. It turns out that the MIP-based heuristic results in fairer schedules, i.e., the distribution of the refused requests is more balanced across the employees. We show that it is beneficial

to run the MIP-based heuristic in an eager manner, i.e., to impose a short time limit for the solution of the subproblems. This setup exploits the fact that optimal solutions of the subproblems are often found within a few seconds, while the majority of time is spent on proving the optimality of this solution. This finding is of general interest in the development of MIP-based heuristics, independent of the context.

Although we develop our approaches for specific real-life problems, they are applicable to other problems discussed in the literature. The decomposition heuristic, the row generation scheme, and the MIP-based heuristic presented in the first, second, and fourth paper, respectively, are easily adaptable to related problems by changing the underlying MIP formulations. In future research, it would be interesting to adapt the MIP-based heuristic to the nurse scheduling problem (cf., e.g., Burke et al., 2004). Furthermore, it would be interesting to adapt the decomposition heuristic and the row generation scheme to the multi-skill project scheduling problem (cf., e.g., Bellenguez-Morineau and Néron, 2007) and the mode identity resource constrained project scheduling problem (cf., e.g., Salewski et al., 1997).

Bibliography

Artigues, C., Koné, O., Lopez, P., Mongeau, M., 2015. Mixed-integer linear programming formulations. In: Schwindt, C., Zimmermann, J. (Eds.), Handbook on Project Management and Scheduling Vol. 1. Springer, Cham, pp. 17–41.

Bellenguez-Morineau, O., Néron, E., 2007. A branch-and-bound method for solving multi-skill project scheduling problem. RAIRO-Operations Research-Recherche Opérationnelle 41 (2), 155–170.

Bixby, R., Rothberg, E., 2007. Progress in computational mixed integer programminga look back from the other side of the tipping point. Annals of Operations Research 149 (1), 37–41.

Bixby, R. E., 2012. A brief history of linear and mixed-integer programming computation. Documenta Mathematica Extra Volume ISMP (2012), 107–121.

Burke, E. K., De Causmaecker, P., Berghe, G. V., Van Landeghem, H., 2004. The state of the art of nurse rostering. Journal of Scheduling 7 (6), 441–499.

Grüter, J., Trautmann, N., Zimmermann, A., 2014. An MBLP model for scheduling assessment centers. In: Huisman, D., Louwerse, I., Wagelmans, A. (Eds.), Operations Research Proceedings 2013. Springer, Berlin, pp. 161–167.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., et al., 2011. MIPLIB 2010. Mathematical Programming Computation 3 (2), 103–163.

Koné, O., Artigues, C., Lopez, P., Mongeau, M., 2011. Event-based MILP models for resource-constrained project scheduling problems. Computers & Operations Research 38 (1), 3–13.

Kopanos, G. M., Kyriakidis, T. S., Georgiadis, M. C., 2014. New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. Computers & Chemical Engineering 68, 96–106.

Lodi, A., 2010. Mixed integer programming computation. In: Jünger, M., Liebling, M. T., Naddef, D., Nemhauser, L. G., Pulleyblank, R. W., Reinelt, G., Rinaldi, G., Wolsey, A. L. (Eds.), 50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art. Springer, Berlin, Heidelberg, pp. 619–645.

Möhring, R. H., 1984. Minimizing costs of resource requirements in project networks subject to a fixed completion time. Operations Research 32 (1), 89–120.

Rihm, T., Trautmann, N., Zimmermann, A., 2016. MIP formulations for an application of project scheduling in human resource management. Flexible Services and Manufacturing Journal, in press.

Salewski, F., Schirmer, A., Drexl, A., 1997. Project scheduling under resource and mode identity constraints: model, complexity, methods, and application. European Journal of Operational Research 102 (1), 88–110.

Vielma, J. P., 2015. Mixed integer linear programming formulation techniques. SIAM Review 57 (1), 3–57.

Williams, H. P., 2013. Model building in mathematical programming. John Wiley & Sons.

Zimmermann, A., Trautmann, N., 2014. Scheduling of assessment centers: an application of resource-constrained project scheduling. In: Fliedner, T., Kolisch, R., Naber, A. (Eds.), Proceedings of the 14th International Conference on Project Management and Scheduling. Munich, pp. 263–266.

Zimmermann, A., Trautmann, N., 2015. A list-scheduling approach for the planning of assessment centers. In: Hanzálek, Z., Kendall, G., McCollum, B., Šůcha, P. (Eds.), Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Application. Prague, pp. 541–554.

Paper I

A decomposition heuristic for short-term planning of assessment centres[1]

Tom Rihm Norbert Trautmann

Department of Business Administration
University of Bern

Contents

[1]Rihm, T., Trautmann, N. (2017). A decomposition heuristic for short-term planning of assessment centres. *European Journal of Industrial Engineering 11(6), 725–751.* DOI: 10.1504/EJIE.2017.10006711.

Abstract

In an assessment centre, several candidates for a job vacancy perform a set of predefined tasks while being observed and evaluated by so-called assessors. For the organizers of such assessment centres, a challenging job is to schedule the tasks and assign the prescribed number of assessors to the tasks such that the total waiting time for the assessors is minimized. This planning situation has been reported to us by a human resource management service provider. Application-specific restrictions distinguish this problem from related scheduling problems discussed in the literature, e.g., the resource-constrained project scheduling problem. We present a mixed-integer programming-based decomposition heuristic, which iterates between pre-scheduling, assignment, and re-scheduling subproblems. Our computational results demonstrate that this novel heuristic outperforms the state-of-the-art approaches on a set of 240 benchmark instances. Furthermore, this heuristic provides optimal solutions to a set of four real-life instances.

1.1 Introduction

Human capital is considered to be a key success factor for many companies and organizations (cf., e.g., Hitt et al., 2001; Skaggs and Youndt, 2004). To recruit the best candidates for job vacancies, a large number of companies operate assessment centres (cf., e.g., Lievens and Thornton III, 2005; Melchers et al., 2010). The objective of such an assessment centre (AC) is to systematically evaluate the skills and abilities of the candidates using tasks that are frequently encountered in the vacant position (cf., e.g., Collins et al., 2003). According to Spector et al. (2000), typical examples of such tasks are project presentations, in-basket exercises, structured interviews, and role-play exercises. While performing the tasks, the candidates are observed and evaluated by so-called assessors. Because these assessors are generally high-level officials of the company (e.g., senior managers) or trained specialists (e.g., psychologists), ACs are relatively expensive (cf., e.g., Wirz et al., 2013). Hence, minimizing the assessors' waiting times is of particular importance during the planning of ACs.

We consider the short-term planning of such assessment centres. This problem has been reported to us by a human resource management service provider that organizes ACs for companies on a regular basis. The assessment centre planning problem (ACP) consists of scheduling a set of predefined tasks and a lunch break for each candidate and

of assigning the prescribed number of assessors and actors to these tasks. The assigned assessors observe and evaluate how the candidates perform the tasks. The assigned actors are required for role-play exercises, e.g. playing the role of a subordinate or of an unhappy customer. The tasks may include some preparation time for the candidate at the beginning and some evaluation time for the assessor(s) and the actor(s) at the end. Furthermore, the lunch breaks need to be scheduled within a prescribed time window. Two assessor-assignment rules have to be considered. First, the number of different assessors assigned to a candidate at least once must lie between given lower and upper bounds. The lower bound targets a fair and objective overall evaluation of each candidate, whereas the upper bound aims to reduce time-consuming discussions among assessors. Second, there may be candidates and assessors who know each other personally. In such a case, the assessor must not observe the candidate (no-go relationship). The objective of the ACP is to minimize the total waiting time for the assessors. Because the assessors meet before the start and after the completion of all tasks and lunch breaks, this objective corresponds to minimizing the total duration of the AC.

The ACP can be interpreted as an extension of the resource-constrained project scheduling problem (RCPSP): each candidate's tasks and lunch break correspond to project activities, and the candidates, assessors, and actors represent renewable resources. The ACP, however, does not contain precedence relationships among the activities, but extends the RCPSP with the two above-described assessor-assignment rules. Rihm et al. (2016) adapted five basic mixed-integer linear programming (MIP) formulations that were developed for the RCPSP to the ACP. Their computational results indicated that only small-sized instances (20–30 activities) were solved to optimality within reasonable computation times. In contrast, the comparative study of Kopanos et al. (2014) showed for the RCPSP that a wide range of medium- and large-sized instances (30–90 activities) were solved to optimality using the basic MIP formulations. These results indicate that besides the lack of precedence relationships, the two above-described assessor-assignment rules have a strong impact on the computation time of general-purpose solvers. Hence, we propose to consider the scheduling decisions and the assignment decisions separately.

In this paper, we propose a decomposition heuristic that divides the ACP into a pre-scheduling, an assignment, and a re-scheduling subproblem. Each of the three subproblems is solved using an appropriate MIP formulation. In the pre-scheduling subproblem, the assessor-assignment rules are dropped. We model this subproblem as an RCPSP. Because the assessor-assignment rules are dropped, a feasible assessor assignment may not exist for the schedule obtained. In the assignment subproblem, the assessors are assigned to the activities such that so-called assessor conflicts are minimized, i.e., the total

time during which one assessor is assigned to more than one activity is minimized. We model this subproblem as an extension of the generalized graph colouring problem. In the re-scheduling subproblem, the assessor conflicts are resolved by re-scheduling some activities, while the assessor assignments are maintained. To improve the obtained feasible solution, an improvement routine applies the decomposition heuristic multiple times while exploiting information about the current best feasible solution. An useful characteristic of the proposed decomposition is that an optimal solution of the pre-scheduling subproblem corresponds to a strong lower bound for the overall problem. In a computational analysis, we apply this decomposition heuristic to four real-life instances and to 240 systematically generated benchmark instances. Our results demonstrate that the proposed decomposition heuristic outperforms the state-of-the-art approaches in terms of solution quality and lower bounds. Furthermore, this heuristic provides optimal solutions to the four real-life instances.

The remainder of this paper is organized as follows. In Section 1.2, we review the related literature. In Section 1.3, we depict the best-performing MIP formulation for the ACP. We use this formulation as the basis for solving the re-scheduling subproblem. In Section 1.4, we present our novel decomposition heuristic and illustrate the heuristic with an example. In Section 1.5, we report the design and the results of our computational analysis. In Section 1.6, we provide some concluding remarks and discuss possible directions for future research.

1.2 Related literature

In Section 1.2.1, we summarize the existing solution approaches for the ACP. In Section 1.2.2, we review related planning problems that contain scheduling and assignment decisions similar to the ACP. In Section 1.2.3, we provide an overview of decomposition heuristics for this type of planning problems.

1.2.1 ACP

The ACP discussed in this paper was first described in Grüter et al. (2014). They proposed an MIP formulation to solve this problem. In this formulation, each activity is split into several sub-activities to model the preparation, execution, and evaluation times. Rihm et al. (2016) interpreted the ACP as an extension of the RCPSP and analysed the performance of two discrete-time (DT) formulations and three continuous-time (CT) formulations for the ACP. For a comprehensive overview of different MIP formulations for the RCPSP, we refer to Artigues et al. (2015). In DT formulations, the activities can

only start or end at predefined points in time. Typically, such formulations involve time-indexed variables. Conversely, in CT formulations, the activities can start at any point in time. The two DT formulations of Rihm et al. (2016) are based on pulse variables (cf. Pritsker et al., 1969) and on/off variables (cf. Kopanos et al., 2014), respectively. The three CT formulation include assessor-assignment variables, resource-flow variables (cf. Artigues et al., 2003), and overlapping variables (cf. Kopanos et al., 2014), respectively, to model the resource constraints. However, despite the recent improvements in optimization software and computer hardware (cf., e.g., Bixby, 2012; Koch et al., 2011; Lodi, 2010), only small-sized instances can be solved to optimality within reasonable computation times using these MIP formulations. For practical applications, the performance of these MIP formulations is insufficient.

To address this drawback, Zimmermann and Trautmann (2015) developed a multi-pass list scheduling heuristic for the ACP. Under this heuristic, the activities are (a) ordered in a list using a priority rule and (b) scheduled sequentially using a schedule-generation scheme. Steps (a) and (b) are executed multiple times; in each iteration, the order of the activities in the list is varied by applying random sampling. This list scheduling heuristic provides good feasible solutions in short computation times; however, there is still a considerable average relative deviation of the objective function values obtained from the best known lower bounds.

1.2.2 Related planning problems

The ACP consists of an assignment subproblem and a scheduling subproblem. In this section, we review planning problems discussed in the literature that also consist of these two subproblems. However, none of these planning problems contain all of the problem characteristics of the ACP.

The multi-mode resource-constrained project scheduling problem (MRCPSP) consists of scheduling a set of project activities to be executed in a specific mode subject to precedence relationships and limited availability of renewable and non-renewable resources (cf., e.g., Hartmann and Briskorn, 2010; Mika et al., 2015). The selected mode determines the duration and the resource requirements of an activity. The objective is to minimize the project duration. In the ACP, the candidates' tasks and lunch breaks correspond to project activities. The candidates, assessors, and actors represent renewable resources, and each feasible assignment of assessors to an activity corresponds to a different execution mode. Because the tasks are unrelated, the ACP does not contain precedence relationships or non-renewable resources. In contrast, the MRCPSP does not contain specific rules to select the modes analogous to the assessor-assignment rules.

The multi-skill project scheduling problem (MSPSP) is a variant of the MRCPSP. In this problem, the renewable resources are multi-skilled employees (cf., e.g., Bellenguez-Morineau and Néron, 2007). Each activity requires a prescribed number of employees with predefined skills, and each employee can perform at most one activity at a time. Each combination of employees that satisfies the activity's skill requirements corresponds to a feasible way to execute that activity. Bounds for the workload of each employee are considered. The ACP can be viewed as an extension of the MSPSP. Each candidate and the set of assessors and actors can each be interpreted as a subset of employees who have the same skills. Assessors with no-go relationships correspond to employees who lack the appropriate skill to observe certain candidates, and the assignment of alternative employees corresponds to alternative assessor assignments. Extensions of the MSPSP are considered by, e.g., Drezet and Billaut (2008) and Li and Womer (2009). However, an equivalent to the assessor-assignment rules is not considered in the MSPSP or its extensions.

Assignment and scheduling decisions are also encountered within other real-life applications. Typical examples include the scheduling of batch process operations in the chemical industry (cf., e.g., Blömer and Günther, 2000; Maravelias, 2006; Reklaitis, 1996), the course and examination timetabling (cf., e.g., Carter and Laporte, 1996, 1998; Dorneles et al., 2014; Schaerf, 1999), the planning and scheduling of operating rooms (cf., e.g., Cardoen et al., 2010; Jebali et al., 2006), and the scheduling of technicians and tasks (cf., e.g., Cordeau et al., 2010; Zamorano and Stolletz, 2017).

1.2.3 Decomposition approaches

A large variety of heuristic decomposition approaches have been proposed for large-scale optimization (cf., e.g., Ball, 2011; Zanakis et al., 1989). The main idea of decomposition approaches is to decompose an initial large and complex problem into smaller and easier subproblems, which can be solved within reasonable computation times. Depending on the relation between the subproblems, two types of decompositions are distinguished (cf., e.g., Zanakis et al., 1989). In the first type, each subproblem is solved independently. The resulting solutions are combined into a solution of the overall problem. In the second type, the subproblems are solved in a given order. Thus, the solution of a subproblem is required as an input for the consecutive subproblem. The solution of the last subproblem corresponds to a solution of the overall problem.

For the MRCPSP and its extensions, the second type of decomposition is the most commonly used (cf., e.g., Ballestín et al., 2013; De Reyck and Herroelen, 1999; Toffolo et al., 2016). In this case, a mode is first assigned to each activity, and subsequently, the

starting times of the activities are determined. These subproblems are solved using different methods. Ballestín et al. (2013) proposed a simulated annealing and an evolutionary algorithm for these two steps, respectively. De Reyck and Herroelen (1999) used a tabu-search method, and Toffolo et al. (2016) applied different MIP formulations to solve the subproblems. Serafini and Speranza (1994) applied a similar decomposition: first, a mode is assigned to each activity; second, a sequence of activities is assigned to each resource; and third, the starting times of the activities are determined. Furthermore, they added a feedback loop to iteratively solve the three subproblems. In this loop, critical activities are selected for which the mode assignment is revised in the subsequent iteration.

All these decomposition approaches are appropriate for the MRCPSP because the mode affects the duration of the activities and hence the total duration of the project. For the ACP, however, the duration of a task is constant, regardless of the assessors assigned. Assigning the assessors before scheduling the tasks can lead to poor solutions. Thus, we propose a decomposition heuristic that proceeds in reverse order.

1.3 Assessment centre planning problem

In this section, we present the MIP formulation for the ACP that performs the best in the comparative analysis of Rihm et al. (2016). The notation that we use throughout this paper is summarized in Tables 1.1 and 1.2.

This MIP formulation includes continuous variables S_i to model the start times of the activities $i \in I$. For activities that include a preparation time, the candidate starts with the preparation at time S_i. Figure 1.1 shows at which time during the execution of an activity the candidate, the assessor(s), and the actor(s) are required. During the preparation time, only the candidate is present. The assessor(s) and the actor(s) join the candidate immediately after the preparation is completed, i.e. at time $S_i + p_i^C$. During the evaluation time, only the assessor(s) and the actor(s) are present, discuss their observations, and evaluate the candidate. This evaluation time can differ between the assessor(s) and the actor(s). Because the candidates are primarily evaluated by the assessors, the evaluation time of the actors does not exceed the evaluation time of the assessors, i.e., $p_i^A \geq p_i^P$ always applies. Due to fairness and objectivity considerations, no waiting times are allowed between the preparation, execution, and evaluation times. A waiting time for a candidate would increase his/her preparation time, whereas a waiting time for the assessors and actors could bias their evaluations of the candidate.

Because each candidate must perform the same tasks and take a lunch break, the assignment of the candidates to the activities is given by the sets I_c ($c \in C$). For the

Table 1.1: Sets and parameters of the MIP formulations

A	Set of assessors
C	Set of candidates
E	Set of edges of the conflict graph
I	Set of activities (including lunch breaks)
I^A, I^P	Set of activities that require assessors (I^A) and actors (I^P)
I_a^A	Set of activities for which the respective candidate has a no-go relationship with assessor a
I_c	Set of activities that require candidate $c \in C$
I_K	Set of activities that require one of the randomly selected candidates
I^L	Set of lunch breaks
I^{tabu}	Set of activities in the tabu list
N	Set of candidate-assessor pairs (c, a) with a no-go relationship
P	Set of actors
ES_i, LS_i	Earliest (ES_i) and latest (LS_i) start times for activity i
M	Sufficiently large number
p_i	Total duration of activity i (including preparation and execution times)
p_i^C	Preparation time of activity i for candidates
p_i^A, p_i^P	Evaluation time of activity i for assessors (p_i^A) and actors (p_i^P)
r_i^A, r_i^P	Number of assessors (r_i^A) and actors (r_i^P) required by activity i
T	Upper bound on the duration of the assessment centre (i.e., length of a day)
L, U	Lower (L) and upper (U) bounds on the number of assessors that are assigned to a candidate at least once (assessor-assignment rule)
w_{ij}	Weight of edge $(i, j) \in E$

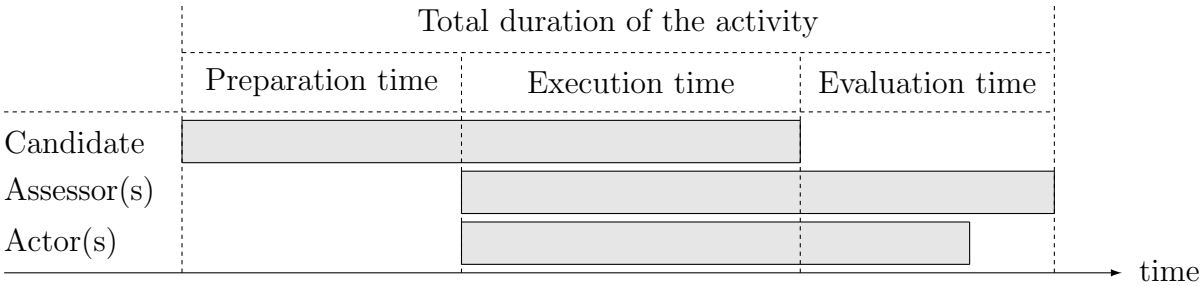


Figure 1.1: Illustration of preparation, execution, and evaluation times

Table 1.2: Variables of the MIP formulations

D	AC duration
S_i	Start time of activity i for the candidate
V_{ca}	$= 1$, if assessor a is assigned to candidate c at least once $= 0$, otherwise
W_{ij}	$= 1$, if activity i and j are assigned to the same assessor $= 0$, otherwise
X_{it}	$= 1$, if activity i starts at time point t $= 0$, otherwise
Y_{ij}^C	$= 1$, if activity i is performed before $j > i$ by a candidate $= 0$, otherwise
Y_{ij}^A	$= 1$, if activity i is performed before $j \neq i$ by the assessors $= 0$, otherwise
Y_{ij}^P	$= 1$, if activity i is performed before $j \neq i$ by the actors $= 0$, otherwise
Z_{ia}^A	$= 1$, if assessor a is assigned to activity i $= 0$, otherwise
Z_{ip}^P	$= 1$, if actor p is assigned to activity i $= 0$, otherwise

assessors and the actors, the assignment to the activities is performed by the model with the binary assignment variables Z_{ia}^A and Z_{ip}^P, respectively. To ensure that all activities that require the same candidate, assessor, or actor do not overlap, binary sequencing variables (Y_{ij}^C, Y_{ij}^P, and Y_{ij}^P) are used. A distinction must be made between candidates, assessors, and actors because of the preparation and evaluation times. Finally, binary variables V_{ca} are used to model the assessor-assignment rules, i.e., $V_{ca} = 1$ if assessor a is assigned to candidate c at least once.

The objective is to minimize the total waiting time for the assessors, i.e., the total duration D of the AC (in the following, we use the term AC duration):

$$\text{Min } D$$

This duration D corresponds to the latest completion time of an activity; see con-

straints (1.1).

$$D \geq S_i + p_i \qquad (i \in I) \tag{1.1}$$

Constraints (1.2)–(1.5) link the start time variables to the sequencing variables. Constraints (1.2) and (1.3) ensure that the activities that require the same candidate do not overlap. Because candidate c is not required during the evaluation time, activity j can start p_i^A time units before the completion of activity i at the earliest.

$$S_j \geq S_i - M + (p_i - p_i^A + M)Y_{ij}^C \qquad (c \in C, i, j \in I_c : i < j) \tag{1.2}$$
$$S_i \geq S_j - M + (p_j - p_j^A + M)(1 - Y_{ij}^C) \qquad (c \in C, i, j \in I_c : i < j) \tag{1.3}$$

Constraints (1.4) and (1.5) guarantee that the activities which require the same assessor or actor do not overlap.

$$S_j \geq S_i - M + (p_i - p_j^C + M)Y_{ij}^A \qquad (i, j \in I^A : i \neq j) \tag{1.4}$$
$$S_j \geq S_i - M + (p_i - p_i^A + p_i^P - p_j^C + M)Y_{ij}^P \qquad (i, j \in I^P : i \neq j) \tag{1.5}$$

Constraints (1.6) require that the lunch breaks are taken during the prescribed time window. For the other activities ($i \in I \backslash I^L$), no time-window restrictions exist.

$$ES_i \leq S_i \leq LS_i \qquad (i \in I^L) \tag{1.6}$$

Constraints (1.7) and (1.8) ensure that the required numbers of assessors and actors are assigned to each activity, respectively.

$$\sum_{a \in A} Z_{ia}^A = r_i^A \qquad (i \in I^A) \tag{1.7}$$
$$\sum_{p \in P} Z_{ip}^P = r_i^P \qquad (i \in I^P) \tag{1.8}$$

Constraints (1.9) and (1.10) link the assignment variables to the sequencing variables. If the same assessor a or the same actor p is assigned to two activities i and j, then either activity i is performed before j or j is performed before i.

$$Y_{ij}^A + Y_{ji}^A \geq Z_{ia}^A + Z_{ja}^A - 1 \qquad (i, j \in I^A, a \in A : i < j) \tag{1.9}$$
$$Y_{ij}^P + Y_{ji}^P \geq Z_{ip}^P + Z_{jp}^P - 1 \qquad (i, j \in I^P, p \in P : i < j) \tag{1.10}$$

Constraints (1.11) and (1.12) forbid cycles in the sequencing decisions.

$$Y^A_{ij} + Y^A_{ji} \leq 1 \qquad (i,j \in I^A : i < j) \qquad (1.11)$$

$$Y^P_{ij} + Y^P_{ji} \leq 1 \qquad (i,j \in I^P : i < j) \qquad (1.12)$$

Constraints (1.13) enforce that the number of different assessors that are assigned to a candidate at least once lies within the bounds imposed by the assessor-assignment rule. For example, the service provider requires that each candidate is observed by approximately 50% of all assessors.

$$L \leq \sum_{a \in A} V_{ca} \leq U \qquad (c \in C) \qquad (1.13)$$

The number of times that an assessor can observe the same candidate is not limited. Constraints (1.14) link variables V_{ca} to the assignment variables Z^A_{ia}, i.e., $V_{ca} = 1$ if and only if assessor a is assigned to at least one activity that requires candidate c.

$$\sum_{i \in I_c \setminus I^L} \frac{Z^A_{ia}}{|I_c \setminus I^L|} \leq V_{ca} \leq \sum_{i \in I_c \setminus I^L} Z^A_{ia} \qquad (c \in C, a \in A) \qquad (1.14)$$

Eventually, constraints (1.15) model the no-go relationships.

$$V_{ca} = 0 \qquad ((c,a) \in N) \qquad (1.15)$$

In sum, formulation (MP) reads as follows.

$$\text{(MP)} \quad \begin{cases} \text{Min } D & \\ \text{s.t. } (1.1)\text{–}(1.15) & \\ \quad D \in \mathbb{N}_0 & \\ \quad S_i \geq 0 & (i \in I) \\ \quad Y^C_{ij} \in \{0,1\} & (c \in C,\ i,j \in I_c :\ i < j) \\ \quad Y^A_{ij} \in \{0,1\} & (i,j \in I^A :\ i \neq j) \\ \quad Y^P_{ij} \in \{0,1\} & (i,j \in I^P :\ i \neq j) \\ \quad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\ \quad Z^A_{ia} \in \{0,1\} & (i \in I^A,\ a \in A) \\ \quad Z^P_{ip} \in \{0,1\} & (i \in I^P,\ p \in P) \end{cases}$$

For a more detailed description, we refer to Rihm et al. (2016).

1.4 Decomposition heuristic

In this section, we present our decomposition heuristic for the ACP in detail. For overview purposes, the heuristic is summarized as a flowchart in Figure 1.2. To construct a feasible solution, pre-scheduling, assignment, and re-scheduling subproblems are solved consecutively using appropriate MIP formulations. To improve the resulting solution, the improvement routine includes a modification step to diversify the search process, followed by the same decomposition procedure. However, in contrast to the construction routine, an MIP-based local search heuristic is applied to the pre-scheduling subproblem. The improvement routine is executed until one of the following two stopping criteria is met: (a) a predefined computation time limit is reached, or (b) the solution's objective function value of the re-scheduling subproblem is equal to the lower bound provided by the pre-scheduling subproblem of the construction routine. Because a lower bound of the pre-scheduling subproblem corresponds to a lower bound of the overall problem, the current solution is optimal if stopping criterion (b) takes effect.

In a preliminary version of the decomposition heuristic (cf. Rihm and Trautmann, 2016), we used the pre-scheduling subproblem to obtain a sequencing of the activities. Based on this sequencing, the scheduling and assignment decisions were performed simultaneously. In this paper, we extend the preliminary version as follows: (a) we include additional constraints in the pre-scheduling subproblem to enhance the performance of the heuristic, as shown in constraints (1.21)–(1.25) below; (b) we extend the decomposition by a third subproblem such that the assessor-assignment and the final scheduling decisions are taken separately; (c) we include an improvement routine; and (d) we provide an MIP-based local search heuristic to solve the pre-scheduling subproblem.

In Sections 1.4.1, 1.4.2, and 1.4.3, we describe the pre-scheduling, assignment, and re-scheduling subproblems, respectively. In Section 1.4.4, we present the improvement routine. In Section 1.4.5, we illustrate the decomposition heuristic using an example.

1.4.1 Pre-scheduling subproblem

The pre-scheduling subproblem is a relaxation of the ACP obtained by dropping the assessor-assignment rules. Thus, an optimal solution to this subproblem corresponds to a lower bound for the ACP. Without the assessor assignment rules, all assessors are considered to be identical, and this subproblem can be interpreted as an RCPSP. Thus, each candidate's tasks and lunch break correspond to a project activity. Each candidate

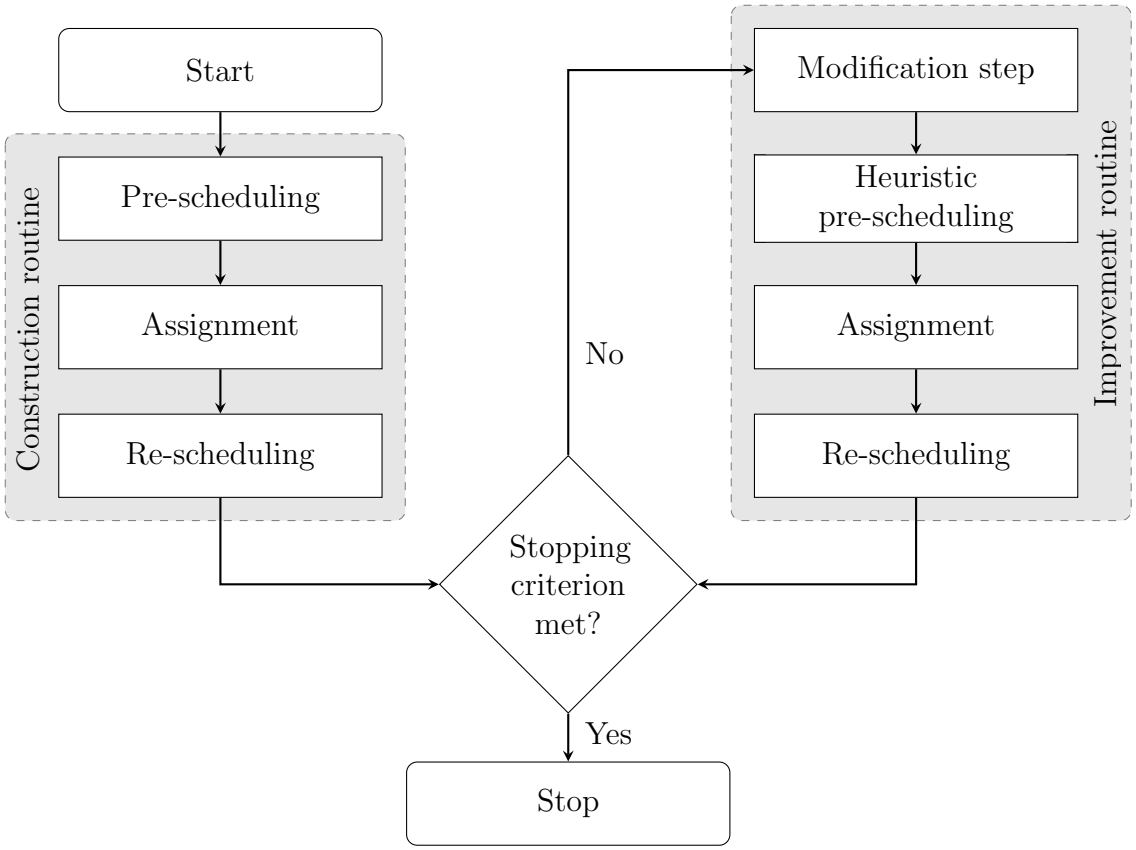


Figure 1.2: Overview of decomposition heuristic

is interpreted as a renewable resource with capacity 1. The set of all assessors (actors) coincide with one renewable resource with a capacity that equals the number of assessors (actors). Because the tasks are unrelated, this pre-scheduling subproblem does not contain precedence relationships.

To solve the pre-scheduling subproblem, we adapt the MIP formulation of Pritsker et al. (1969) for the RCPSP as follows. The start times of the activities are modelled with binary time-indexed variables, i.e. $X_{it} = 1$ if and only if activity i starts at time t. The AC duration is modelled with an integer variable D, which has to be minimized:

$$\text{Min } D$$

Constraints (1.16) state that this duration is greater than or equal to the largest completion time of an activity. The earliest and latest start times (ES_i and LS_i) are required to model the time windows for the lunch break activities ($i \in I^L$). For the other activities

($i \in I \backslash I^L$), we set $ES_i = 0$ and $LS_i = T$.

$$D \geq \sum_{t=ES_i}^{LS_i} (t + p_i) X_{it} \qquad (i \in I) \tag{1.16}$$

Constraints (1.17) ensure that each activity is scheduled once.

$$\sum_{t=ES_i}^{LS_i} X_{it} = 1 \qquad (i \in I) \tag{1.17}$$

Constraints (1.18)–(1.20) cover the resources that represent the candidates, assessors, and actors, respectively. Constraints (1.18) prevent activities that require the same candidate from being executed at the same time. Because the evaluation time for the assessors and actors is included in the total duration, activity i is performed at time t by a candidate if the activity started between time $t + 1 - (p_i - p_i^A)$ and t.

$$\sum_{i \in I_c} \sum_{s=\max(ES_i, t+1-(p_i-p_i^A))}^{\min(LS_i,t)} X_{is} \leq 1 \qquad (c \in C;\ t = 0, \ldots, T) \tag{1.18}$$

Constraints (1.19)–(1.20) ensure that only activities that do not require more than the available number of assessors and actors are executed at the same time. Therefore, activity i is performed at time t by an assessor or actor if the activity started between time $t + 1 - p_i$ and $t - p_i^C$ or time $t + 1 - (p_i - p_i^A + p_i^P)$ and $t - p_i^C$, respectively.

$$\sum_{i \in I^A} \sum_{s=\max(ES_i, t+1-p_i)}^{\min(LS_i, t-p_i^C)} r_i^A X_{is} \leq |A| \qquad (t = 0, \ldots, T) \tag{1.19}$$

$$\sum_{i \in I^P} \sum_{s=\max(ES_i, t+1-(p_i-p_i^A+p_i^P))}^{\min(LS_i, t-p_i^C)} r_i^P X_{is} \leq |P| \qquad (t = 0, \ldots, T) \tag{1.20}$$

To enhance the performance of the heuristic, we include constraints (1.21)–(1.25). These constraints exclude some solutions to the pre-scheduling subproblem for which no feasible assessor assignment exists. Constraints (1.21) state that all activities that are executed at the same time t and for which the corresponding candidate has a no-go relationship

with assessor a require at most $|A| - 1$ assessors.

$$\sum_{i \in I_a^A} \sum_{s=\max(ES_i, t+1-d_i)}^{\min(LS_i, t-p_i^C)} r_i^A X_{i,s} \leq |A| - 1 \qquad (t \in T;\ a \in A) \tag{1.21}$$

Constraints (1.22) widen this principle to all activities that have a no-go relationship with the same two assessors a and e.

$$\sum_{i \in I_{a1}^A \cap I_{a2}^A} \sum_{s=\max(ES_i, t+1-d_i)}^{\min(LS_i, t-p_i^C)} r_i^A X_{i,s} \leq |A| - 2 \qquad (t \in T;\ a1, a2 \in A : a1 \neq a2) \tag{1.22}$$

Constraints (1.23)–(1.25) correspond to lower bounds for the ACP, which have been introduced in Rihm et al. (2016). The objective of these lower bounds is to stop the solver if a solution with a duration equal to a lower bound is found, even though a shorter schedule may exist for the relaxed problem. Constraint (1.23) ensures that the duration D is greater than or equal to the average workload of the assessors rounded up. The shortest preparation time of an activity is added because the assessors cannot start before that time.

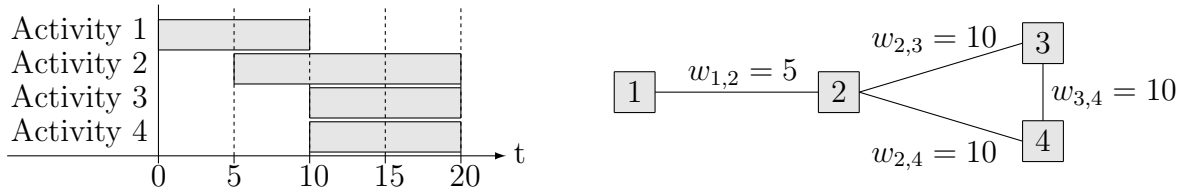
$$D \geq \left\lceil \sum_{i \in I^A} \frac{r_i^A (p_i - p_i^C)}{|A|} \right\rceil + \min_{i \in I^A} p_i^C \tag{1.23}$$

The lower bound of constraint (1.24) is obtained by considering only the activities that require two assessors. The total workload of these activities is evenly distributed among an even number of assessors.

$$D \geq \left\lceil \sum_{i \in I^A : r_i^A = 2} \frac{p_i - p_i^C}{\left\lfloor \frac{|A|}{2} \right\rfloor} \right\rceil + \min_{i \in I^A} p_i^C \tag{1.24}$$

All activities that require the same candidate c must be performed sequentially, i.e., the AC duration cannot be smaller than the total duration of these activities for the candidate:

$$D \geq \max_{c \in C} \left(\sum_{i \in I_c} (p_i - p_i^A) + \min_{i \in I_c} (p_i^A) \right) \tag{1.25}$$



(a) Schedule of the activities for the assessors

(b) Corresponding conflict graph

Figure 1.3: Construction of a conflict graph

In sum, formulation (PS) reads as follows:

$$\text{(PS)} \quad \begin{cases} \text{Min } D & \\ \text{s.t. } (1.16)\text{–}(1.25) & \\ \quad D \in \mathbb{N}_0 & \\ \quad X_{it} \in \{0,1\} & (i \in I;\ t = ES_i, \ldots, LS_i) \end{cases}$$

1.4.2 Assignment subproblem

Given the schedule provided by the pre-scheduling subproblem, the assignment subproblem consists of assigning the required number of assessors to the activities such that the assessor-assignment rules are fulfilled. Because a feasible assignment may not exist for the given schedule, the objective is to minimize the total time during which some assessors are assigned to several activities simultaneously (assessor conflict).

To solve the assignment subproblem, we construct an edge-weighted conflict graph $G = (I, E, w)$. For each activity, one node is created. Two nodes are connected by an edge if the corresponding activities overlap in the schedule (for the assessors). The weight w_e of edge e is equal to the time during which the corresponding activities overlap.

Figure 1.3 illustrates the construction of a conflict graph with an example, which includes four activities. Figure 1.3(a) provides a schedule of the activities for the assessors, and Figure 1.3(b) provides the corresponding conflict graph. Activities 1 and 2 overlap during 5 time units; therefore, the conflict graph contains an edge between nodes 1 and 2 with edge weight $w_{1,2} = 5$. Analogously, the graph contains edges between nodes 2 and 3, 2 and 4, and 3 and 4 with edge weights $w_{2,3} = w_{2,4} = w_{3,4} = 10$.

The assignment subproblem consists of assigning exactly r_i^A assessors to each activity (node) i such that the assessor-assignment rules are met. The objective is to minimize the total weight of the edges in the conflict graph between pairs of nodes for which at least one identical assessor is assigned to both nodes. This subproblem can be considered

as an extension of the generalized graph colouring problem (cf. Carlson and Nemhauser, 1966; Vredeveld and Lenstra, 2003). The generalized graph colouring problem consists of, given the total number of different colours, assigning one colour to each node of the graph. The objective is to minimize the total weight of the monochromatic edges (i.e., edges that have end points with the same colour). In the assignment subproblem, the colours corresponds to the assessors. The assignment subproblem extends the generalized graph colouring problem by the assignment of a pre-defined number of colours to each node (multicolouring, cf. Halldórsson and Kortsarz, 2004), and by the assessor-assignment rules.

To solve the assignment subproblem, we propose a novel MIP formulation. In this formulation, we use the binary conflict variables W_{ij}, which are equal to one if at least one identical assessor is assigned to the activities i and j, $(i,j) \in E$. Furthermore, we use the assessor-assignment variables Z_{ia} and V_{ca} from formulation (MP). The objective is to minimize the total time of assessor conflicts.

$$\text{Min} \sum_{(i,j)\in E} w_{ij} W_{ij}$$

Constraints (1.26) state that an assessor conflict occurs ($W_{ij} = 1$) if the same assessor is assigned to two adjacent nodes in the conflict graph.

$$W_{ij} \geq Z_{ia} + Z_{ja} - 1 \qquad ((i,j) \in E,\ a \in A) \tag{1.26}$$

Constraints (1.27) ensure that the required number of assessors is assigned to each activity.

$$\sum_{a\in A} Z_{ia} = r_i^A \qquad (i \in I^A) \tag{1.27}$$

Constraints (1.28) link the assignment variables Z_{ia} to the binary variables V_{ca}. Consequently, $V_{ca} = 1$ if and only if assessor a is assigned to at least one activity that requires candidate c.

$$\sum_{i\in I_c \setminus I^L} \frac{Z_{ia}}{|I_c \setminus I^L|} \leq V_{ca} \leq \sum_{i\in I_c \setminus I^L} Z_{ia} \qquad (c \in C,\ a \in A) \tag{1.28}$$

Constraints (1.29) and (1.30) take the assessor-assignment rules into account. Constraints (1.29) ensure that the number of assessors assigned to each candidate lies within

the bounds imposed by the assessor-assignment rule.

$$L \leq \sum_{a \in A} V_{ca} \leq U \qquad (c \in C) \tag{1.29}$$

Constraints (1.30) model the no-go relationships.

$$V_{ca} = 0 \qquad ((c,a) \in N) \tag{1.30}$$

In sum, formulation (AS) reads as follows.

$$\text{(AS)} \quad \begin{cases} \text{Min} \displaystyle\sum_{(i,j) \in E} w_{ij} W_{ij} & \\ \text{s.t.} \quad (1.26)\text{–}(1.30) & \\ \qquad V_{ca} = 0 & ((c,a) \in N) \\ \qquad Z^A_{ia} \in \{0,1\} & (i \in I^A,\ a \in A) \\ \qquad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\ \qquad W_{ij} \in \{0,1\} & (i,j \in I) \end{cases}$$

1.4.3 Re-scheduling subproblem

Given the assessor assignment provided by the solution of the assignment subproblem, the re-scheduling subproblem consists of re-scheduling the activities to resolve all assessor conflicts. Furthermore, the actors are assigned to the activities. Thus, a feasible solution for the ACP is constructed.

To solve the re-scheduling subproblem, we adapt formulation (MP) of Section 1.3 as follows. The assignment variables Z^A_{ia} are fixed to the values obtained in the assignment subproblem. Consequently, constraints (1.7) and (1.13)–(1.15) are omitted. In sum,

formulation (RS) reads as follows:

$$
\text{(RS)} \quad \begin{cases}
\text{Min } D & \\
\text{s.t.} \quad (1.1)\text{–}(1.6) & \\
\qquad (1.8)\text{–}(1.12) & \\
\qquad D \in \mathbb{N}_0 & \\
\qquad S_i \geq 0 & (i \in I) \\
\qquad Y^C_{ij} \in \{0,1\} & (c \in C,\ i,j \in I_c :\ i < j) \\
\qquad Y^A_{ij} \in \{0,1\} & (i,j \in I^A :\ i \neq j) \\
\qquad Y^P_{ij} \in \{0,1\} & (i,j \in I^P :\ i \neq j) \\
\qquad Z^P_{ip} \in \{0,1\} & (i \in I^P,\ p \in P)
\end{cases}
$$

In contrast to the pre-scheduling subproblem (discrete-time formulation), the re-scheduling subproblem is solved using a continuous-time formulation. The reason is that in the pre-scheduling subproblem, the assessors are considered as a single resource with a capacity that is equal to the total number of assessors. In this case, discrete-time formulations with time-indexed variables perform the best. In the re-scheduling subproblem, each assessor is considered as a single resource with a unit capacity. In this case, continuous-time formulations with sequencing variables perform the best.

1.4.4 Improvement routine

In the improvement routine, a modification step is executed before the three subproblems are resolved. The goal of this modification step is to use some information about the current ACP solution in the following pre-scheduling subproblem to diversify the solution process. In the modification step, (a) the current best solution for the ACP is selected, (b) an activity $i^* \in I \backslash I^{tabu}$ with the largest time gap to its immediate predecessor (for the same candidate) is chosen, and (c) the earliest start time ES_{i^*} of activity i^* is set to its current start time S_{i^*}. The tabu set I^{tabu} ensures that another activity ($i^* \notin I^{tabu}$) is selected in the following modification step if the solution was not improved in the previous iteration. By setting the earliest start time of activity i^* to its current start time in the schedule, one of the activities scheduled after i^* may be scheduled earlier during the solution process of the following pre-scheduling subproblem. After the modification step, the pre-scheduling, assignment, and re-scheduling subproblems are solved consecutively. Finally, the tabu set I^{tabu} is updated, i.e., activity i^* is added to the tabu set I^{tabu} if the solution was not improved. The lunch breaks ($i \in I^L$) are included in the tabu set I^{tabu}

because the lunch breaks must be scheduled within a narrow prescribed time window. The improvement routine stops if (a) a predefined computation time limit is reached, or (b) the objective function value of the solution to the re-scheduling subproblem is equal to the lower bound provided by the pre-scheduling subproblem of the construction routine, or (c) the tabu set I^{tabu} contains all activities $i \in I$. Algorithm 1.1 describes this improvement routine.

Algorithm 1.1 Improvement routine

Input: ACP solution (after the construction routine) with objective function value D
Tabu list $I^{tabu} \leftarrow I^L$
Lower bound LB computed in the construction routine
Output: ACP solution
while time limit not met **and** $D \neq LB$ **and** $I^{tabu} \neq I$ **do**
Set current solution to best-known ACP solution
$i^* \leftarrow$ activity $i \in I \backslash I^{tabu}$ with largest gap to its predecessor
$ES_{i^*} \leftarrow S_{i^*}$
Apply heuristic pre-scheduling
$ES_{i^*} \leftarrow 0$
Solve assignment subproblem using formulation (AS)
Solve re-scheduling subproblem using formulation (RS)
if ACP solution is improved **then**
$I^{tabu} \leftarrow I^L$
else
$I^{tabu} \leftarrow I^{tabu} \cup \{i^*\}$
end if
end while

In general, the pre-scheduling subproblem requires the most computation time. To reduce the computation time of the pre-scheduling subproblem in the improvement routine, we (a) propose an MIP-based local search heuristic for this subproblem, and (b) use the current best overall solution for the ACP as an initial solution for the heuristic. In the construction routine, we continue to solve the pre-scheduling subproblem exactly to obtain a strong lower bound for the ACP.

The pre-scheduling subproblem is solved using an MIP-based local search heuristic (see Algorithm 1.2). The basic idea of this heuristic is to iteratively improve an initial solution by re-scheduling some activities within narrow time windows. The resulting sub-subproblems are smaller and easier to solve. The sub-subproblems are constructed as follows. For the activities I_K of k randomly selected candidates, the time windows are not restricted. The sequence of these activities can be changed without restrictions. For the remaining activities $i \in I \backslash (I^L \cup I_K \cup \{i^*\})$, the time window is defined such that the activities can be advanced or delayed by at most m time units. The time window of the

Algorithm 1.2 Heuristic pre-scheduling

Input: Current best solution for the ACP and activity i^* of modification step
Output: A solution for the pre-scheduling subproblem
while no stopping criterion met **do**
  $K \leftarrow$ Set of k randomly selected candidates
  for $i \in I \backslash (I^L \cup \{i^*\})$ **do**

$$ES_i \leftarrow \begin{cases} = 0, & \text{if } i \in I_K \backslash I^L; \\ = S_i - m, & \text{if } i \in I \backslash I_K. \end{cases}$$

$$LS_i \leftarrow \begin{cases} = D, & \text{if } i \in I_K \backslash I^L; \\ = S_i + m, & \text{if } i \in I \backslash I_K. \end{cases}$$

  end for
  Solve pre-scheduling subproblem using formulation (PS)
  Update S_i for all $i \in I$
end while

lunch break activities I^L is not changed because this time window is prescribed by the problem instance.

Parameters m and k control the size of the resulting sub-subproblem and thus the computation time. A small value of both parameters leads to small sub-subproblems that can be solved in short computation times. However, the possible improvements are limited. Larger improvements can be obtained for larger values of m and k, but at the expense of more computation time. In the computational analysis, we use $m = 5$ and $k = 3$.

We use two stopping criteria for the MIP-based local search heuristic: (a) a predefined computation time limit is reached, and (b) the solution's objective value is equal to the lower bound derived in the construction routine.

1.4.5 Illustrative example

In this section, we illustrate the decomposition heuristic with an example. We consider an AC with three candidates (C_1, C_2, and C_3), four assessors (A_1, A_2, A_3, and A_4), and two actors (P_1 and P_2). The two assessor-assignment rules are a) each candidate must be observed by at least $L = 2$ and at most $U = 3$ different assessors, and b) candidate C_3 must never be observed by assessor A_2 because of a no-go relationship, i.e., $N = \{(C_3,A_2)\}$.

Each of the three candidates has to perform three tasks and one lunch break. Thus, the problem consists of 12 activities, $I = \{1, 2, \ldots, 12\}$. Table 1.3 shows the main data of the activities. Row 2 lists all activities that correspond to one of the three tasks or to a lunch break. Rows 3 to 5 state which activity is associated with which task and candidate. For example, activities 1 to 3 refer to task 1. Activity 1 corresponds to task 1

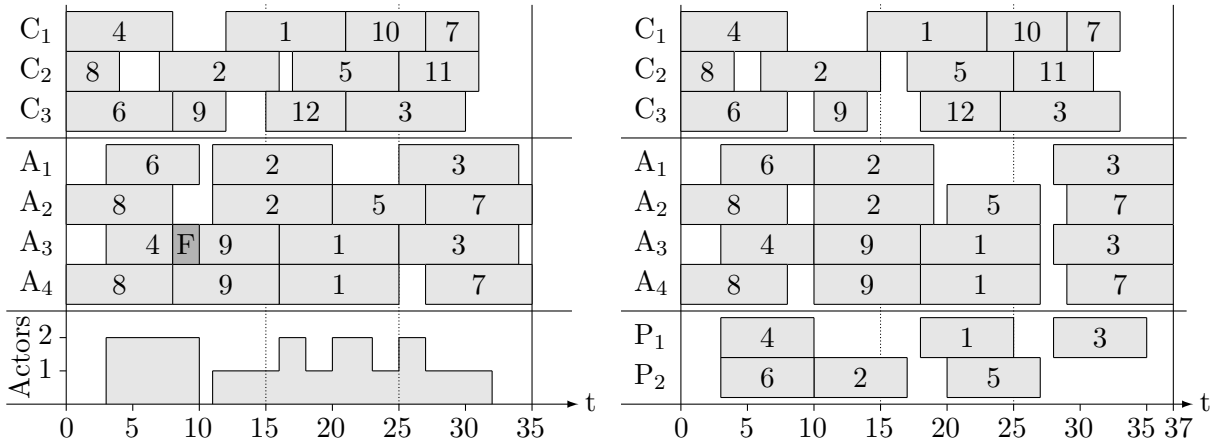
Table 1.3: Illustrative example: main data of the activities

	Task 1	Task 2	Task 3	Lunch break
Corresponding activity i	1,2,3	4,5,6	7,8,9	10,11,12
Activity related to candidate C_1	1	4	7	10
Activity related to candidate C_2	2	5	8	11
Activity related to candidate C_3	3	6	9	12
Duration p_i	13	10	8	6
Preparation time p_i^C	4	3	0	0
Execution time	5	5	4	6
Evaluation time for assessors p_i^A	4	2	4	0
Evaluation time for actors p_i^P	2	2	0	0
Earliest start time ES_i	0	0	0	15
Latest start time LS_i	50	50	50	25
Required number of assessors r_i^A	2	1	2	0
Required number of actors r_i^P	1	1	0	0

for candidate 1, activity 2 corresponds to task 1 for candidate 2, and so forth. The duration, the preparation and evaluation times, the earliest and latest start times, and the required number of assessors and actors of the activities are shown in rows 6 to 14. Here, one time unit corresponds to 5 minutes.

We applied the proposed decomposition heuristic to this illustrative example. Figure 1.4 shows two temporary schedules in the course of the construction routine. The dotted lines indicate the earliest and latest start times for the lunch breaks, and the solid line indicates the AC duration. Figure 1.4(a) shows the schedule obtained by solving the pre-scheduling subproblem and the assessor assignment obtained by solving the assignment subproblem. The pre-scheduling subproblem is solved to optimality. Hence, the duration of 35 corresponds to a lower bound for the ACP. The assessor assignment obtained by solving the assignment subproblem is not feasible for the overall problem because assessor A_3 is assigned to two overlapping activities (activities 4 and 9). The corresponding assessor conflict F is highlighted in dark grey. Notably, due to the absence of specific actor-assignment rules, the pre-scheduling subproblem ensures that a feasible assignment of the actors to the activities exists. However, the actors are only assigned in the re-scheduling subproblem.

Figure 1.4(b) shows the schedule obtained by solving the re-scheduling subproblem. Some activities are delayed to eliminate the assessor conflict. The obtained schedule is



(a) Solution of the pre-scheduling and assignment subproblems

(b) Solution of the re-scheduling subproblem

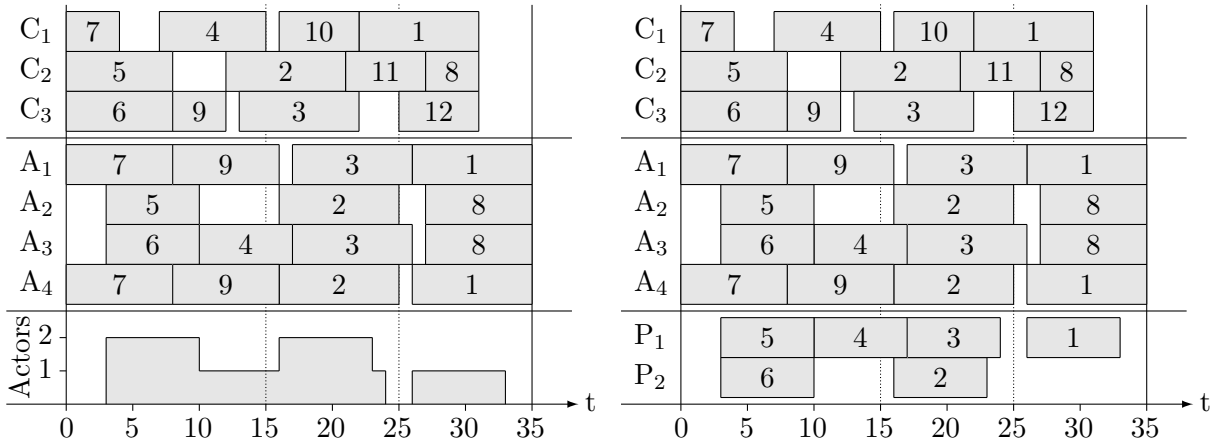
Figure 1.4: Illustrative example: construction routine

feasible for the overall ACP, and the duration is 37.

In the improvement routine, the schedule of Figure 1.4(b) is improved as follows. In the modification step, activity 1 is selected. Figure 1.5(a) provides the results of the following pre-scheduling and assignment subproblems. Due to the modification step, activities 10 and 7 are shifted before activity 1. The duration of the schedule is 35, and the assignment no longer contains assessor conflicts. Figure 1.5(b) shows the final solution, including the actor assignments. This solution is optimal because the duration is equal to the lower bound provided by the pre-scheduling subproblem in the construction routine.

1.5 Computational analysis

In this section, we compare the performance of the proposed decomposition heuristic (DH) to the performance of the best-performing MIP formulation (MP) of Rihm et al. (2016) as stated in Section 1.3 and to the performance of the list scheduling heuristic (LSH) of Zimmermann and Trautmann (2015). In Section 1.5.1, we describe the test instances used. In Section 1.5.2, we present the design of the analysis. In Section 1.5.3, we report and analyse the computational results. By varying the time limits for the different subproblems, the focus of the decomposition heuristic can be put either on computing strong lower bounds (pre-scheduling subproblem) or on improving the current solutions (assignment and re-scheduling subproblem). In Section 1.5.4, we assess the performance of the heuristic if shorter overall time limits were considered, i.e., if the focus is put on finding good feasible solutions quickly.



(a) Solution of the pre-scheduling and assignment subproblems

(b) Solution of the re-scheduling subproblem

Figure 1.5: Illustrative example: improvement routine

1.5.1 Test instances

We use two different test sets for the computational analysis. The first set contains the four real-life instances presented by Grüter et al. (2014), and the second set contains the 240 systematically generated benchmark instances generated by Rihm et al. (2016). For all instances, the upper bound on the duration of the assessment centre T is set to 200; this value is prescribed by the human resource provider. The earliest and the latest start times for the lunch break activities $i \in I^L$ are set to $ES_i = 30$ and $LS_i = 78$, respectively. For the other activities $i \in I \backslash I^L$, we set $ES_i = 0$ and $LS_i = T$. Here, 1 time unit corresponds to 5 minutes. The lower and upper bounds (L and U) of the assessor-assignment rule are defined such that each candidate is observed by approximately 50% of all assessors:

$$L = \left\lfloor \frac{|A|}{2} \right\rfloor \quad \text{and} \quad U = \left\lceil \frac{|A|}{2} \right\rceil + 1.$$

The data of the four real-life instances is summarized in Table 1.4. The instances consist of between 6 and 11 candidates, 9 to 11 assessors, 2 or 3 actors, and between 36 and 66 activities. The last column indicates whether at least one no-go relationship exists.

The 240 generated benchmark instances were constructed by varying the following five complexity factors. The experimental levels of each complexity factor are based on real-life data.

- The number of candidates $n^C \in \{4, 5, \ldots, 10, 11\}$.

Table 1.4: Real-life instances

Instance	Number of candidates	Number of assessors	Number of actors	Number of activities	No-go relationships
RL1	7	10	2	42	no
RL2	11	11	3	66	no
RL3	9	11	3	54	yes
RL4	6	9	3	36	no

- The number of tasks $n^E \in \{4, 5\}$. The tasks were randomly selected from a set of 15 real-life tasks. Each of these tasks has a predefined duration, predefined preparation and evaluation times, and requires a predefined number of assessors and actors.
- The average number of assignments per assessor $a^S \in \{6.0, 8.5, 10.4\}$. The number of assessors of an instance depends on both the specific tasks selected and the value of a^S. Thus, the number of assessors n^A is equal to the nearest integer to $\sum_{i \in I^A} r_i^A / a^S$.
- The ratio of assessors who have at least one no-go relationship $a^N \in \{\frac{1}{6}, \frac{1}{3}\}$.
- The average number of no-go relationships per assessor with at least one no-go relationship $a^R \in \{2, 3\}$. The no-go relationships were randomly assigned to pairs of candidates and assessors such that (a) the ratio of assessors who have at least one no-go relationship a^N is met and (b) at least $\lfloor n^A/2 \rfloor$ different assessors can be assigned to each candidate.

The number of actors was set to 3 for all instances. The test set contains one instance for each combination of the complexity factor levels ($8 \cdot 2 \cdot 3 \cdot 2 \cdot 2 = 192$ benchmark instances). Additionally, the test set contains $8 \cdot 2 \cdot 3 = 48$ benchmark instances without no-go relationships (i.e., $a^N = a^R = 0$).

1.5.2 Experimental design

We implemented the decomposition heuristic (DH) and the MIP formulation (MP) in AMPL, and we used the Gurobi Optimizer 7.0.1 with the default solver settings as the solver. The list scheduling heuristic (LSH) was implemented in Java. All computations were performed on a workstation equipped with two 6-core Intel(R) Xeon(R) X5650 CPUs running at 2.66 GHz, and with 24 GB RAM. The computations were performed using all available CPU cores. For the four real-life instances, we prescribed a computation time

Table 1.5: Time limits used by the decomposition heuristic [in s]

Instance	Subproblem	Construction routine	Improvement routine
Real-life instances	Pre-scheduling	1,800	30/10
	Assignment	3	3
	Re-scheduling	10	10
Generated instances	Pre-scheduling	300	30/10
	Assignment	3	3
	Re-scheduling	10	10

limit of 3,600 seconds. For the 240 benchmark instances, we prescribed a computation time limit of 1,200 seconds.

In addition to the overall time limit, we used different time limits to solve the subproblems of the decomposition heuristic (see Table 1.5). These time limits help to balance between finding good lower bounds and good feasible solutions within the overall time limit. For the 240 benchmark instances, we set the solver time limit to 300 seconds to solve the pre-scheduling subproblem in the construction routine. In the improvement routine, we ran the MIP-based local search heuristic for 30 seconds. Here, we set a time limit of 10 seconds to solve each MIP formulation. To solve the assignment and the re-scheduling subproblem, we set a solver time limit of 3 and 10 seconds, respectively. For each subproblem, the best feasible solution found is returned if the time limit is reached. If no feasible solution is found for a subproblem within the time limit, the decomposition heuristic terminates and returns the best feasible solution found so far (if any).

1.5.3 Computational results

The results for the four real-life instances are shown in Table 1.6. For each approach, we report the objective function value (OFV), the lower bound (LB) if present, and the required computation time in seconds (CPU). For MP, we report the lower bounds obtained by the solver during the branch-and-bound process. For DH, we report the lower bounds obtained by the solver during the branch-and-bound process of the pre-scheduling subproblem. For each instance, the best OFV obtained are highlighted in boldface. Our decomposition heuristic solved all four real-life instances to optimality. Both approaches, MP and LSH, provided an optimal solution to instance RL4 only. However, they were not able to prove that their solution obtained is optimal within the prescribed computation

Table 1.6: Numerical results for real-life instances

Instance	MP			LSH		DH		
	OFV	LB	CPU	OFV	CPU	OFV	LB	CPU
RL1	88	69	3,603	86	3,600	**82**	82	88
RL2	138	62	3,600	113	3,600	**110**	110	31
RL3	107	62	3,603	96	3,600	**90**	90	1,249
RL4	**82**	68	3,603	**82**	3,600	**82**	82	46

time limit.

The results for the 240 benchmark instances are summarized in Table 1.7. For each approach, we state the average relative deviation of the solutions to the lower bounds obtained by the decomposition heuristic ($\varnothing\Delta_{\text{LB}}$), the total number of optimal solutions ($\#OPT$), and the total number of best solutions ($\#BEST$). To determine the number of optimal solutions, we compare the objective function value obtained with the lower bound obtained with DH. The number of best solutions corresponds to the number of times that a method generates a best solution. Furthermore, we report the average relative MIP gap ($\varnothing\Delta_{\text{MIP}}$) obtained by the solver for model MP. We compare the average results for different levels of complexity factors. For each complexity factor level, the best results are highlighted in boldface.

Overall, the three approaches are able to find a feasible solution for each instance within the prescribed computation time. For each level of the different complexity factors and for each performance criterion, the decomposition heuristic provides the best results. Over all instances, the average gap $\varnothing\Delta_{\text{LB}}$ of DH is 0.64%. Furthermore, DH solves 166 out of 240 instances to optimality and provides a best solution for 237 out of 240 instances. Because the average MIP gap $\varnothing\Delta_{\text{MIP}}(= 41.5\%)$ is considerably higher than the average gap $\varnothing\Delta_{\text{LB}}(= 7.9\%)$ for MP, we conclude that the lower bounds of the DH are much stronger than the lower bounds of MP. Indeed, for each instance, the lower bound of DH is equal to or stronger than the lower bound obtained with MP within the limited computation time.

Next, we study the impact of the complexity parameters. Depending on the number of activities $|I| = n^C(n^E + 1)$, we divide the instances into three groups. Instances with 20–30 activities are considered small sized, instances with 31–50 activities are considered medium sized, and instances with 51–66 activities are considered large sized. For the small-sized instances, MP performs better than LSH and almost as good as DH. However,

Table 1.7: Factor-dependent results for benchmark instances

Complexity		Number of	$\varnothing\Delta_{\text{MIP}}[\%]$	$\varnothing\Delta_{\text{LB}}[\%]$			$\#OPT$			$\#BEST$		
	factor	instances	MP	MP	LSH	DH	MP	LSH	DH	MP	LSH	DH
$\|I\|$	20–30	75	31.5	0.6	1.1	**0.3**	48	38	**60**	60	44	**73**
	31–50	105	42.8	6.3	3.2	**0.5**	8	15	**74**	8	23	**104**
	51–66	60	51.6	19.8	5.7	**1.3**	0	0	**32**	0	0	**60**
a^S	6.0	80	26.6	7.0	3.5	**0.3**	15	6	**64**	15	6	**80**
	8.5	80	44.9	10.2	4.1	**1.1**	15	18	**50**	20	23	**79**
	10.4	80	52.9	6.6	2.0	**0.6**	26	29	**52**	33	38	**78**
a^N	0.00	48	42.3	8.4	3.2	**0.8**	11	12	**33**	13	15	**48**
	0.17	96	41.3	7.8	3.2	**0.6**	22	20	**67**	28	27	**95**
	0.33	96	41.2	7.8	3.1	**0.6**	23	21	**66**	27	25	**94**
a^R	0	48	42.3	8.4	3.2	**0.8**	11	12	**33**	13	15	**48**
	2	96	41.3	7.7	3.2	**0.6**	23	21	**68**	28	25	**93**
	3	96	41.2	7.9	3.2	**0.6**	22	20	**65**	27	27	**96**
All instances		240	41.5	7.9	3.2	**0.6**	56	53	**166**	68	67	**237**

the performance of MP is affected the most by the size of the instances. For the large-sized instances, the average gap $\varnothing\Delta_{\text{LB}}[\%]$ increases to 19.8% for MP, but only to 1.3% for DH.

The complexity factor a^S affects the three methods differently. DH finds the most optimal solutions for the lowest value of a^S. In contrast, MP and LSH find more optimal solutions for high values than for lower values of a^S. The performance differences for DH can be explained as follows. If the average number of assignments per assessor a^S is low, then there are relatively many assessors compared to the total number of activities. In this case, it is more likely that a feasible assessor assignment exists for a solution of the pre-scheduling subproblem. Hence, DH performs better. Complexity factors a^N and a^R do not affect the solution quality of the three methods. The number of optimal solutions is lower for $a^N = 0$ and $a^R = 0$, but there are less instances with these factor levels (48 instances for $a^R = 0$ compared to 96 instances for $a^R = 2$ and $a^R = 3$, respectively).

Table 1.8 lists for the set of 240 instances the results of the construction routine, which stops if the first feasible solution is found. Furthermore, this table reports the intermediate results after 600, 900, and 1,200 seconds. In addition to the already mentioned performance criteria $\varnothing\Delta_{\text{LB}}[\%]$ and $\#OPT$, we state the average number of iterations

Table 1.8: Intermediate results of decomposition heuristic

Intermediate step	$\varnothing\Delta_{\mathrm{LB}}[\%]$	$\#OPT$	$\varnothing\#IT$
After construction routine	2.48	134	0
After 600 s	0.80	162	5.5
After 900 s	0.71	163	8.6
After 1200 s	0.64	166	11.4

performed in the improvement routine of DH ($\varnothing\#IT$). The average gap $\varnothing\Delta_{\mathrm{LB}}$ of the solutions provided by the construction routine is 2.48%. Here, 134 instances are solved to optimality. After 600 seconds, the average gap is reduced to 0.80%. In this time, the improvement routine performed on average 5.5 iterations. During the next 600 seconds, the average gap is reduced further. The final gap is 0.64%.

In total, the decomposition heuristic is able to prove the optimality of 166 instances out of 240. For the remaining 74 instances, either the solution does not correspond to an optimal solution or the solution is optimal but the lower bound provided by the pre-scheduling subproblem is not tight. Indeed, many examples exist for which the optimal objective function value of the pre-scheduling subproblem is strictly smaller than the optimal objective function value of the overall problem. However, our computational results demonstrate that the lower bound provided by the pre-scheduling subproblem is on average better than the lower bound obtained with MP within the limited computation time.

1.5.4 Computational results for shorter time limits

A salient characteristic of the decomposition heuristic is that each lower bound of the pre-scheduling subproblem corresponds also to a lower bound for the ACP. To exploit this characteristic, in Section 1.5.3, we used a large time limit to solve the pre-scheduling subproblem. The results of this section indicate that the decomposition heuristic remains competitive in terms of solution quality when shorter overall time limits are considered. We ran the heuristic with overall time limits of 15, 30, 60, 120, and 300 seconds, respectively. Due to the reduction of the overall time limit, the time limits to solve the individual subproblems of the decomposition heuristic have also been adapted; we set the time limits to 5 seconds to solve each subproblem. Additionally, we set the parameter MIPFocus of the Gurobi solver to 1. This parameter determines the MIP solution strategy of the solver. When this parameter is set to 1, Gurobi focuses on quickly generating

Table 1.9: Comparison of LSH and DH for different overall time limits

Time	$\varnothing\Delta_{\text{LB}}[\%]$		$\#OPT$		$\#BEST$	
limit [in s]	LSH	DH	LSH	DH	LSH	DH
15	4.14	**4.07**	43	**55**	137	**153**
30	3.92	**2.91**	45	**72**	115	**180**
60	3.75	**2.06**	46	**94**	100	**206**
120	3.61	**1.59**	50	**112**	93	**221**
300	3.42	**1.22**	52	**133**	87	**229**

good feasible solutions rather than increasing the lower bound.

Table 1.9 compares the results of LSH and DH for the 240 benchmark instances. Notably, the overall time limits considered are too short for the MIP formulation (MP) to find competitive solutions. The same criteria as in Table 1.7 are used to evaluate the two approaches. The gap Δ_{LB} is calculated based on the same lower bounds as in Section 1.5.3. For each overall time limit considered, the best results are highlighted in boldface.

For an overall time limit of 15 seconds, the performance of the two approaches is approximately the same. However, if the overall time limits increase, the performance (in terms of the three criteria) of DH improves much stronger. For an overall time limit of 30 seconds or more, DH considerably outperforms LSH.

1.6 Conclusions and outlook

In this paper, we considered a real-life assessment centre planning problem. The problem consists of scheduling a set of predefined tasks for each candidate and of assigning the prescribed number of assessors and actors to these tasks. The objective is to minimize the total waiting time for the assessors. We proposed a mathematical programming-based heuristic under which the initial problem is iteratively decomposed into pre-scheduling, assignment, and re-scheduling subproblems. An advantage of this decomposition is that an optimal solution of the pre-scheduling subproblem corresponds to a lower bound for the overall problem. In a comparative analysis, we showed that the heuristic provides better solutions and better lower bounds than the state-of-the-art methods. In particular, the heuristic is able to provide optimal solutions to a set of four real-life benchmark instances in limited computation time. Moreover, by varying the time limits to solve the different

subproblems, the heuristic is well scalable with respect to an overall computation time limit.

The optimal solutions generated by our decomposition heuristic are used to obtain insights into the structure of optimal solutions. In this way, the decomposition heuristic has already contributed to improving the performance of the list scheduling heuristic that is currently used by the service provider.

In future research, the development of an exact tailored solution approach (e.g., a branch-and-cut approach) is necessary to completely close the optimality gaps. Furthermore, the real-life context of assessment centres gives rise to other interesting variants of the planning problem. This includes the planning of assessment centres that last more than one day, the consideration of so-called group tasks that are performed simultaneously by multiple candidates, and the minimization of the required number of assessors rather than the total waiting time.

Bibliography

Artigues, C., Koné, O., Lopez, P., Mongeau, M., 2015. Mixed-integer linear programming formulations. In: Schwindt, C., Zimmermann, J. (Eds.), Handbook on Project Management and Scheduling Vol. 1. Springer, Cham, pp. 17–41.

Artigues, C., Michelon, P., Reusser, S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149 (2), 249–267.

Ball, M. O., 2011. Heuristics based on mathematical programming. Surveys in Operations Research and Management Science 16 (1), 21–38.

Ballestín, F., Barrios, A., Valls, V., 2013. Looking for the best modes helps solving the MRCPSP/max. International Journal of Production Research 51 (3), 813–827.

Bellenguez-Morineau, O., Néron, E., 2007. A branch-and-bound method for solving multi-skill project scheduling problem. RAIRO-Operations Research-Recherche Opérationnelle 41 (2), 155–170.

Bixby, R. E., 2012. A brief history of linear and mixed-integer programming computation. Documenta Mathematica Extra Volume ISMP (2012), 107–121.

Blömer, F., Günther, H.-O., 2000. LP-based heuristics for scheduling chemical batch processes. International Journal of Production Research 38 (5), 1029–1051.

Cardoen, B., Demeulemeester, E., Beliën, J., 2010. Operating room planning and scheduling: A literature review. European Journal of Operational Research 201 (3), 921–932.

Carlson, R., Nemhauser, G., 1966. Scheduling to minimize interaction cost. Operations Research 14 (1), 52–58.

Carter, M. W., Laporte, G., 1996. Recent developments in practical examination timetabling. In: Burke, E. K., Ross, P. (Eds.), Practice and Theory of Automated

Timetabling: Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling, Edinburgh, 1995. Springer, Berlin, Heidelberg, pp. 3–21.

Carter, M. W., Laporte, G., 1998. Recent developments in practical course timetabling. In: Burke, E. K., Carter, M. (Eds.), Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling, Toronto, 1997. Springer, Berlin, Heidelberg, pp. 3–19.

Collins, J. M., Schmidt, F. L., Sanchez-Ku, M., Thomas, L., McDaniel, M., Le, H., 2003. Can basic individual differences shed light on the construct meaning of assessment center evaluations? International Journal of Selection and Assessment 11 (1), 17–29.

Cordeau, J.-F., Laporte, G., Pasin, F., Ropke, S., 2010. Scheduling technicians and tasks in a telecommunications company. Journal of Scheduling 13 (4), 393–409.

De Reyck, B., Herroelen, W., 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 119 (2), 538–556.

Dorneles, Á. P., de Araújo, O. C., Buriol, L. S., 2014. A fix-and-optimize heuristic for the high school timetabling problem. Computers & Operations Research 52, 29–38.

Drezet, L.-E., Billaut, J.-C., 2008. A project scheduling problem with labour constraints and time-dependent activities requirements. International Journal of Production Economics 112 (1), 217–225.

Grüter, J., Trautmann, N., Zimmermann, A., 2014. An MBLP model for scheduling assessment centers. In: Huisman, D., Louwerse, I., Wagelmans, A. (Eds.), Operations Research Proceedings 2013. Springer, Berlin, pp. 161–167.

Halldórsson, M. M., Kortsarz, G., 2004. Multicoloring: Problems and techniques. In: Fiala, J., Koubek, V., Kratochvíl, J. (Eds.), Mathematical Foundations of Computer Science 2004, Lecture Notes in Computer Science, vol 3153. Springer, Berlin, Heidelberg, pp. 25–41.

Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207 (1), 1–14.

Hitt, M. A., Biermant, L., Shimizu, K., Kochhar, R., 2001. Direct and moderating effects of human capital on strategy and performance in professional service firms: A resource-based perspective. Academy of Management Journal 44 (1), 13–28.

Jebali, A., Alouane, A. B. H., Ladet, P., 2006. Operating rooms scheduling. International Journal of Production Economics 99 (1), 52–62.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., et al., 2011. MIPLIB 2010. Mathematical Programming Computation 3 (2), 103–163.

Kopanos, G. M., Kyriakidis, T. S., Georgiadis, M. C., 2014. New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. Computers & Chemical Engineering 68, 96–106.

Li, H., Womer, K., 2009. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. Journal of Scheduling 12 (3), 281–298.

Lievens, F., Thornton III, G. C., 2005. Assessment centers: Recent developments in practice and research. In: Evers, A., Anderson, N., Voskuijl, O. (Eds.), The Blackwell Handbook of Personnel Selection. Blackwell Publishing Ltd, Malden, pp. 243–264.

Lodi, A., 2010. Mixed integer programming computation. In: Jünger, M., Liebling, M. T., Naddef, D., Nemhauser, L. G., Pulleyblank, R. W., Reinelt, G., Rinaldi, G., Wolsey, A. L. (Eds.), 50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art. Springer, Berlin, Heidelberg, pp. 619–645.

Maravelias, C. T., 2006. A decomposition framework for the scheduling of single-and multi-stage processes. Computers & Chemical Engineering 30 (3), 407–420.

Melchers, K. G., Kleinmann, M., Prinz, M. A., 2010. Do assessors have too much on their plates? The effects of simultaneously rating multiple assessment center candidates on rating quality. International Journal of Selection and Assessment 18 (3), 329–341.

Mika, M., Waligóra, G., Węglarz, J., 2015. Overview and state of the art. In: Schwindt, C., Zimmermann, J. (Eds.), Handbook on Project Management and Scheduling Vol. 1. Springer, Cham, pp. 445–490.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: a zero-one programming approach. Management Science 16 (1), 93–108.

Reklaitis, G. V., 1996. Overview of scheduling and planning of batch process operations. In: Reklaitis, G. V., Sunol, A. K., Rippin, D. W. T., Hortaçsu, Ö. (Eds.), Batch Processing Systems Engineering: Fundamentals and Applications for Chemical Engineering. Springer, Berlin, Heidelberg, pp. 660–705.

Rihm, T., Trautmann, N., 2016. A decomposition approach for an assessment center planning problem. In: Ruiz, R., Alvarez-Valdes, R. (Eds.), Proceedings of the 15th International Conference on Project Management and Scheduling. Valencia, pp. 206–209.

Rihm, T., Trautmann, N., Zimmermann, A., 2016. MIP formulations for an application of project scheduling in human resource management. Flexible Services and Manufacturing Journal, in press.

Schaerf, A., 1999. A survey of automated timetabling. Artificial Intelligence Review 13 (2), 87–127.

Serafini, P., Speranza, M. G., 1994. A decomposition approach for a resource constrained scheduling problem. European Journal of Operational Research 75 (1), 112–135.

Skaggs, B. C., Youndt, M., 2004. Strategic positioning, human capital, and performance in service organizations: A customer interaction approach. Strategic Management Journal 25 (1), 85–99.

Spector, P. E., Schneider, J. R., Vance, C. A., Hezlett, S. A., 2000. The relation of cognitive ability and personality traits to assessment center performance. Journal of Applied Social Psychology 30 (7), 1474–1491.

Toffolo, T. A. M., Santos, H. G., Carvalho, M. A. M., Soares, J. A., 2016. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. Journal of Scheduling 19 (3), 295–307.

Vredeveld, T., Lenstra, J. K., 2003. On local search for the generalized graph coloring problem. Operations Research Letters 31 (1), 28–34.

Wirz, A., Melchers, K. G., Lievens, F., De Corte, W., Kleinmann, M., 2013. Trade-offs between assessor team size and assessor expertise in affecting rating accuracy in assessment centers. Journal of Work and Organizational Psychology 29 (1), 13–20.

Zamorano, E., Stolletz, R., 2017. Branch-and-price approaches for the multiperiod technician routing and scheduling problem. European Journal of Operational Research 257 (1), 55–68.

Zanakis, S. H., Evans, J. R., Vazacopoulos, A. A., 1989. Heuristic methods and applications: a categorized survey. European Journal of Operational Research 43 (1), 88–110.

Zimmermann, A., Trautmann, N., 2015. A list-scheduling approach for the planning of assessment centers. In: Hanzálek, Z., Kendall, G., McCollum, B., Šůcha, P. (Eds.), Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Application. Prague, pp. 541–554.

Paper II

MIP formulations for an application of project scheduling in human resource management[2]

Tom Rihm Norbert Trautmann Adrian Zimmermann

Department of Business Administration
University of Bern

Contents

[2]Rihm, T., Trautmann, N., Zimmermann, A. (2016). MIP formulations for an application of project scheduling in human resource management. *Flexible Services and Manufacturing Journal.* Advance online publication. DOI: 10.1007/s10696-016-9260-8

Abstract

In the literature, various discrete-time and continuous-time mixed-integer linear programming (MIP) formulations for project scheduling problems have been proposed. The performance of these formulations has been analyzed based on generic test instances. The objective of this study is to analyze the performance of discrete-time and continuous-time MIP formulations for a real-life application of project scheduling in human resource management. We consider the problem of scheduling assessment centers. In an assessment center, candidates for job positions perform different tasks while being observed and evaluated by assessors. Because these assessors are highly qualified and expensive personnel, the duration of the assessment center should be minimized. Complex rules for assigning assessors to candidates distinguish this problem from other scheduling problems discussed in the literature. We develop two discrete-time and three continuous-time MIP formulations, and we present problem-specific lower bounds. In a comparative study, we analyze the performance of the five MIP formulations on four real-life instances and a set of 240 instances derived from real-life data. The results indicate that good or optimal solutions are obtained for all instances within short computational time. In particular, one of the real-life instances is solved to optimality. Surprisingly, the continuous-time formulations outperform the discrete-time formulations in terms of solution quality.

2.1 Introduction

Over the past decades, mixed-integer linear programming (MIP) methods have been significantly improved (cf., e.g., Koch et al., 2011; Bixby, 2012) and successfully applied to a large variety of real-life scheduling problems in manufacturing and services. Two major advantages of MIP methods are the flexibility to account for changes in the problem setting and the possibility to obtain upper or lower bounds on the solutions. In general, different formulations can be used to model the same planning problem. Because the performance of MIP approaches is determined by the underlying formulation (cf., e.g., Vielma, 2015), alternative formulations should be considered for each planning problem.

In this paper, we investigate an assessment center planning problem (ACP). This problem was reported to us by a human resource management service provider that organizes

assessment centers (AC) for firms. The goal of an AC is to evaluate some candidates' job-related skills and abilities for one or several open positions (cf., e.g., Collins et al., 2003). In an AC, each candidate performs multiple tasks, and for each task, a prescribed number of assessors (i.e., psychologists or managers) is required. Some tasks involve role play and additionally require a prescribed number of actors. For example, the actors might represent unhappy customers with whom the candidate must interact. Tasks sometimes require a preparation time during which only the candidate is present. During the execution of the task, the candidate is joined by the assessors and the actor. Some tasks include a subsequent evaluation during which the assessors and the actors discuss their observations. This evaluation time can differ between assessors and actors. Each candidate takes a lunch break within a prescribed time window. When assigning assessors to tasks, the following rules must be considered: each candidate should be observed by approximately half the number of assessors; if a candidate and an assessor know each other personally, no observation is allowed, which is called a no-go relationship. Assessors are expensive, and hence, their total waiting time should be minimized. Because the assessors meet before the start and after the completion of all tasks and lunch breaks, this objective corresponds to minimizing the total duration of the AC (in what follows the AC duration for short). The planning problem consists of (1) scheduling all tasks and a lunch break for each candidate and (2) determining which assessors are assigned to which candidate during which task such that the AC duration is minimized.

The ACP can be interpreted as an extension of the resource-constrained project scheduling problem (RCPSP). The RCPSP consists of scheduling a set of activities subject to completion-start precedence and renewable-resource constraints such that the project duration is minimized. For the ACP, each candidate's tasks and lunch break correspond to project activities, and the candidates, assessors, and actors represent renewable resources. However, the ACP does not involve precedence relationships among the activities, but the above-described additional constraints. In the literature, different MIP formulations have been proposed for the RCPSP. In discrete-time (DT) formulations, the planning horizon is divided into a set of time intervals of equal length, and the activities can only start or end at the endpoints of these intervals. Conversely, in continuous-time (CT) formulations, the activities can start at any point in time. The DT formulations usually involve binary time-indexed variables. However, the meaning of these variables differ between the formulations, e.g., so-called pulse variables indicate whether an activity starts or ends at a specific point in time (cf. Pritsker et al., 1969; Christofides et al., 1987; Kopanos et al., 2014), and on/off variables specify whether an activity is in progress at a given time (cf. Kaplan, 1988; Mingozzi et al., 1998; Kopanos et al., 2014). The CT formulations differ

with regard to the modeling of the resource constraints, e.g., Artigues et al. (2003) use resource-flow variables, and Kopanos et al. (2014) use overlapping variables. For a comprehensive overview of different MIP formulations for the RCPSP, we refer to Artigues et al. (2015).

In this paper, we provide two DT formulations and three CT formulations for the ACP. The two DT formulations are based on pulse variables (DT–P) and on/off variables (DT–O), respectively. The three CT formulations use assessor-assignment variables (CT–A), resource-flow variables (CT–F), and overlapping variables (CT–O), respectively, to model the resource constraints. Moreover, we provide problem-specific lower bounds. The different MIP formulations are tested on four real-life instances and 240 test instances based on real-life data. For all instances, good or optimal solutions are obtained within short computational time. In detail, formulation CT–A consistently outperforms the other four formulations in terms of solution quality. However, using DT–P, the best MIP-based lower bounds are obtained. Furthermore, only with DT–P, optimality is proven for one of the real-life instances within the prescribed time limit. Nevertheless, in contrast to the RCPSP, the CT formulations provide better solutions than the DT formulations.

The remainder of this paper is structured as follows. In Section 2.2, we describe the ACP using an illustrative example and relate the ACP to the RCPSP. In Section 2.3, we provide an overview of the related literature. In Section 2.4, we present the MIP formulations for the ACP. In Section 2.5, we derive the problem-specific lower bounds. In Section 2.6, we discuss the design and the results of our comparative analysis. In Section 2.7, we provide some concluding remarks and an outlook on future research.

2.2 Planning problem

In Section 2.2.1, we describe the problem features of the ACP in detail and illustrate them through an example. In Section 2.2.2, we discuss the relation between the ACP and the RCPSP.

2.2.1 Illustration of the planning problem

In our illustrative example, the participants of the AC are as follows: there are three candidates, C_1, C_2 and C_3; four assessors, A_1, A_2, A_3 and A_4; and an actor, P_1. A no-go relationship exists between candidate C_3 and assessor A_2. Each of the three candidates must perform the three tasks E_1, E_2, and E_3, and take a lunch break.

The tasks of the illustrative example are listed in Table 2.1. The durations of the tasks are stated in 5-minute time units. Tasks E_1 and E_3 require two assessors, and task E_2

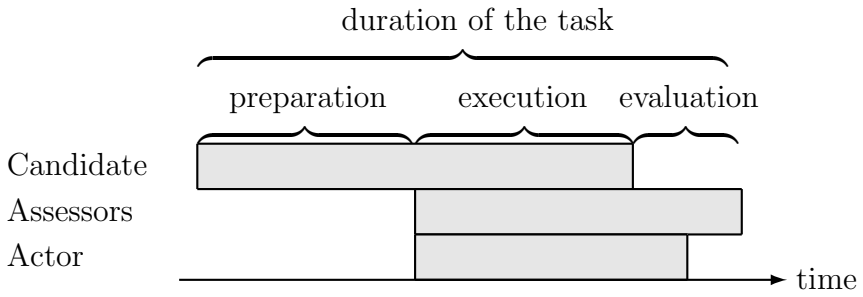


Figure 2.1: Varying requirements for candidate, assessors, and actor during task E_1

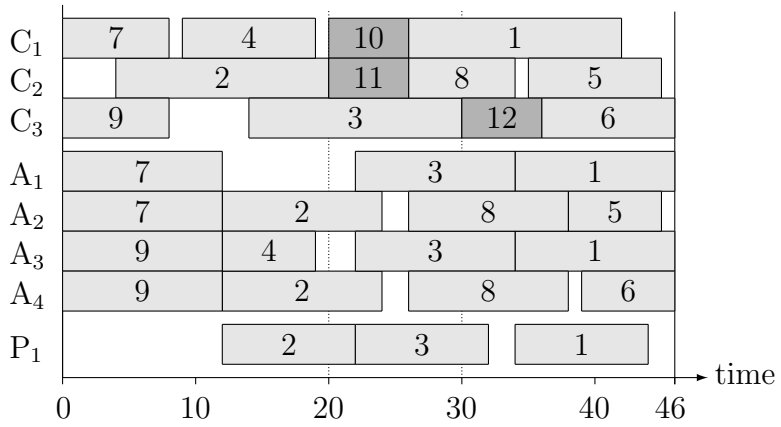
Table 2.1: Tasks of illustrative example

Task	E_1	E_2	E_3
Required number of assessors	2	1	2
Required number of actors	1	-	-
Duration	20	10	12
Duration of preparation time (candidates)	8	3	-
Duration of execution time	8	7	8
Duration of evaluation time (assessors)	4	-	4
Duration of evaluation time (actors)	2	-	-

requires one assessor. Task E_1 involves role play and requires one actor. Tasks E_1 and E_2 include a preparation time, and tasks E_1 and E_3 include an evaluation time. Figure 2.1 shows at which time during the execution of task E_1 the candidate, the assessors, and the actor are required. The evaluation time differs between the assessors and the actor. Due to fairness and objectivity considerations, no waiting times are allowed between the preparation, the execution, and the evaluation. A waiting time for a candidate would increase the preparation time, whereas a waiting time for the assessors and actors could bias their evaluations of the candidate.

The earliest and latest possible start times for the lunch break are 20 and 30, respectively. The duration of the lunch break is 6 time units. Because each candidate has a lunch break and performs each of the three tasks exactly once, a total of 12 activities are considered. Table 2.2 shows the indices of these activities.

The rules for assigning assessors to candidates are as follows: each candidate should be observed by at least half of the total number of assessors rounded down and by at most half of the total number of assessors rounded up plus one. The lower limit ensures an objective overall evaluation for each candidate, and the upper limit is motivated by



Assignments			
	Candidate		
Assessor	C_1	C_2	C_3
A_1	✓		✓
A_2	✓	✓	
A_3	✓		✓
A_4		✓	✓

Figure 2.2: Optimal schedule of the illustrative example (left) and corresponding assessor assignment (right)

fairness considerations. The difference between the upper and lower limits facilitates the assessor assignment without affecting fairness. The number of times that an assessor can observe the same candidate is not limited. In the illustrative example, each candidate must be observed by 2 to 3 different assessors. Additionally, because a no-go relationship exists, candidate C_3 can never be observed by assessor A_2.

An optimal schedule for the illustrative example is presented in Figure 2.2. The dotted lines indicate the earliest and latest start times for the lunch breaks, and the solid line indicates the AC duration. Whether an assessor has been assigned to a candidate at least once is indicated by a checkmark (✓).

Table 2.2: Activity indices of the illustrative example

Candidate	Task			Lunch break
	E_1	E_2	E_3	
C_1	1	4	7	10
C_2	2	5	8	11
C_3	3	6	9	12

2.2.2 Relation to the RCPSP

The ACP includes many problem features of the well-known RCPSP. Both planning problems consider activities that require prescribed amounts of some renewable resources during their execution. In the case of the ACP, the execution of each task and the

lunch break for each candidate correspond to a project activity, and the candidates, assessors, and actors can be interpreted as renewable resources. The ACP does not involve precedence relationships among the activities.

In the RCPSP, only the capacities and not the individual units of the renewable resources are considered. However, in the ACP, the assessor-assignment rules require that all activities that use a particular resource unit can be identified. Therefore, the assessor-assignment rules cannot be formulated in the RCPSP.

If each assessor is interpreted as a renewable resource with unit capacity, then alternative execution modes must be defined in order to represent the alternative assessor assignments. This corresponds to the multi-mode extension of the RCPSP (MRCPSP). Because each candidate must be observed by approximately half the number of the assessors, the assessor assignments interdepend. Such interdependencies between modes are not considered in the MRCPSP. Before assigning any assessors to a candidate, all modes are feasible. However, selecting the modes for some activities causes several of the modes of the other activities to be infeasible.

2.3 Literature review

In Section 2.3.1, we provide an overview of different MIP formulations for the RCPSP which can be used as the basis for MIP formulations of the ACP. In Section 2.3.2, we discuss recent works that focus on comparing MIP formulations for extensions of the RCPSP and for specific real-life problems.

2.3.1 MIP formulations for the RCPSP

In DT formulations, binary time-indexed variables are used that indicate the start, end, or the state (e.g., in progress) of an activity at a specific time. For DT formulations, three types of binary variables can be distinguished (cf. Artigues et al., 2015). Beside the pulse and on/off variables described in the Introduction, there are step variables that indicate whether an activity starts at or before a specific point in time (cf. Klein, 2000; Bianco and Caramia, 2013). Furthermore, Bianco and Caramia (2013) introduce continuous variables that specify the percentage of completion of the activities at each point in time.

In CT formulations, the activities can start or finish at any time rather than at predefined time points such as in DT. Artigues et al. (2003) present a CT formulation based on resource flows. Besides the continuous start-time variables, this formulation requires two additional sets of variables. The first set consists of binary sequencing variables that determine for each pair of activities whether one precedes the other or whether both are

executed in parallel. The second set consists of continuous resource-flow variables for modeling the resource constraints. Kopanos et al. (2014) present another CT formulation with continuous start-time variables, binary sequencing variables, and binary overlapping variables. In combination with the sequencing variables, the overlapping variables are used to model the resource constraints. Other CT formulations are based on events (e.g. Koné et al., 2011) or on minimal forbidden sets (e.g. Alvarez-Valdes and Tamarit, 1993).

For the RCPSP, the performances of these different MIP formulations are compared in Bianco and Caramia (2013), Koné et al. (2011), and Kopanos et al. (2014). They all use generic test instances, which are provided in, e.g., Kolisch and Sprecher (1997) and Vanhoucke et al. (2008). For these test instances, Koné et al. (2011) and Kopanos et al. (2014) show that the performance is primarily affected by the number of activities and the length of the planning horizon. The performances of the DT formulations are negatively affected by the length of the planning horizon because the numbers of variables and constraints depend on the number of time points considered. In contrast, the performances of the CT formulations are negatively affected by the number of activities because the number of sequencing variables increases exponentially with the number of activities. Typically, DT-based formulations are the most competitive and yield the best LP relaxations. However, no formulation consistently dominates the others, as different formulations perform better for different problem settings.

In this study, we adapt different RCPSP formulations such that they can be applied to the ACP. From the DT formulations, we select the RCPSP formulations of Pritsker et al. (1969) and Kopanos et al. (2014). The basic DT formulation of Pritsker et al. (1969) still performs very well compared to newer formulations (cf., e.g., Koné et al., 2011). Kopanos et al. (2014) show that their two DT formulations outperform other DT formulations presented in the literature. Their DT formulations differ with regard to the modeling of the precedence constraints. For the ACP, these two formulations are identical because there are no precedence constraints. From the CT formulations, we adapt the formulations of Artigues et al. (2003) and Kopanos et al. (2014). The CT formulation of Artigues et al. (2003) performs well compared to other CT formulations if there are specific problem characteristics such as long activity durations (cf., e.g., Koné et al., 2011). Kopanos et al. (2014) show that their two CT formulations outperform other CT formulations presented in the literature; we adapted their best-performing CT formulation.

2.3.2 Comparative studies of MIP formulations

In addition to the aforementioned comparative studies of the RCPSP, the performances of alternative MIP formulations have also been compared for various other planning prob-

lems. In the following, we provide an overview of such comparative studies for extensions of the RCPSP and for some real-life problems.

Some extensions of the RCPSP for which alternative MIP formulations have been compared are as follows. In Koné et al. (2013), the performances of alternative DT and CT formulations are compared for an extension of the RCPSP with so-called storage resources. Storage resources are consumed and produced at the project activities' start times and completion times, respectively. As in Koné et al. (2011), the authors conclude that no MIP formulation consistently yields the best results. A comparative performance analysis of alternative DT formulations for the RCPSP with flexible resource profiles is provided in Naber and Kolisch (2014). With flexible resource profiles, the resource utilization of an activity is not constant but rather can be adjusted from period to period. The results of the comparative study in Naber and Kolisch (2014) indicate that an MIP formulation based on Bianco and Caramia (2013) dominates all other DT formulations. In the study of Zapata et al. (2008), alternative DT and CT formulations for the MRCPSP with multiple projects are compared. The authors conclude that the best MIP formulation depends on the specific characteristics of each problem instance.

Comparative analyses have also been conducted for MIP formulations in real-life applications. Stefansson et al. (2011) develop DT and CT formulations for a large-scale production scheduling problem originating from a pharmaceutical producer. In this problem, customers order specific products, which need to be produced in a four-stage production process such that the requested quantity and delivery date of the order are met. The results obtained for eight test instances indicate that the CT formulation obtains better solutions within shorter computational time than the DT formulation. Furthermore, in Chen et al. (2012), a comparative analysis of different mixed-integer nonlinear programming formulations for the scheduling of crude-oil refinement operations is presented. The planning problem includes several processing steps, from unloading marine vessels to producing various crude-oil based products. In a recent study, Ambrosino et al. (2015) evaluated the performance of two alternative MIP formulations for the multi-port master bay plan problem. This problem involves the placement of containers on a containership such that the overall berthing costs of the ship's multi-port journey are minimized.

2.4 MIP formulations for the ACP

In this section, we present our five MIP formulations for the ACP. The notation of the MIP formulations is provided in Tables 2.3 and 2.4. In Section 2.4.1, we present the CT formulation that uses the assessor-assignment decisions to model the resource constraints

(CT–A). In Section 2.4.2, we derive the CT formulation with resource-flow variables (CT–F). In Section 2.4.3, we present the CT formulation with overlapping variables (CT–O). In Sections 2.4.4 and 2.4.5, we present the DT formulation with pulse variables (DT–P) and the DT formulation with on/off variables (DT–O), respectively.

Table 2.3: Sets and parameters of the MIP formulations

C	Set of candidates
A	Set of assessors
P	Set of actors
N	Set of candidate-assessor pairs (c, a) with a no-go relationship
I	Set of activities $i = 1, \ldots, n$ (including lunch breaks)
I_c	Set of activities that require candidate $c \in C$
I^A, I^P	Set of activities that require assessors (I^A) and actors (I^P)
I^L	Set of lunch breaks
ES^L, LS^L	Earliest (ES^L) and latest (LS^L) start time for the lunch breaks
p_i	Duration of activity i
p_i^C	Preparation time of activity i for candidates
p_i^A, p_i^P	Evaluation time of activity i for assessors (p_i^A) and actors (p_i^P)
r_i^A, r_i^P	Number of assessors (r_i^A) and actors (r_i^P) required by activity i
M	Sufficiently large number
T	Upper bound on the duration of the assessment center

2.4.1 Formulation CT–A

In this section, we present the continuous-time formulation that uses the assessor-assignment decisions to model the resource constraints (CT–A). In a preliminary version of this MIP formulation (cf. Grüter et al., 2014), each activity is split into several sub-activities to model the preparation, the execution, and the evaluation times. However, this results in an unnecessary large number of variables and constraints. In the following, we model the ACP without splitting the activities.

We distinguish between three types of resources: candidates, assessors, and actors. Each candidate is modeled as a renewable resource with capacity 1. The set of all assessors (actors) is modeled as one renewable resource with a capacity that equals the number of assessors (actors). Due to the capacity of 1, the resource constraints for the candidates are modeled using binary sequencing variables, i.e., $Y_{ij}^C = 1$ ($Y_{ij}^C = 0$) if activity i (j) is

Table 2.4: Variables of the MIP formulations

D	AC duration
S_i	Start time of activity i for the candidate
X_{it}	$= 1$, if activity i starts at time point t; $= 0$, otherwise.
Y_{ij}^C	$= 1$, if activity i is performed before $j > i$ by a candidate; $= 0$, otherwise.
Y_{ij}^A	$= 1$, if activity i is performed before $j \neq i$ by the assessors; $= 0$, otherwise.
Y_{ij}^P	$= 1$, if activity i is performed before $j \neq i$ by the actors; $= 0$, otherwise.
Z_{ia}^A	$= 1$, if assessor a is assigned to activity i; $= 0$, otherwise.
Z_{ip}^P	$= 1$, if actor p is assigned to activity i; $= 0$, otherwise.
V_{ca}	$= 1$, if assessor a is assigned to candidate c at least once; $= 0$, otherwise.
F_{ij}^C	$= 1$, if a candidate is sent from activity i to j; $= 0$, otherwise.
F_{ij}^A	Number of assessors sent from activity i to j
F_{ij}^P	Number of actors sent from activity i to j
$\widehat{Y}_{ij}$	$= 1$, if activity i starts before or at the same time as j for assessors; $= 0$, otherwise.
O_{ji}^A	$= 1$, if activity j finishes after the start of activity i for assessors; $= 0$ or 1, otherwise.
O_{ji}^P	$= 1$, if activity j finishes after the start of activity i for actors; $= 0$ or 1, otherwise.
W_{it}	$= 1$, if i is processed at time t by the candidates; $= 0$, otherwise.

completed some time before the start of activity j (i) by the corresponding candidate. For the assessors and actors, the resource constraints are modeled using binary sequencing variables (Y_{ij}^A and Y_{ij}^P), and binary assignment variables (Z_{ia}^A and Z_{ip}^P). For the assessors, the sequencing variable Y_{ij}^A is equal to 1 if activity i is completed some time before the start of activity j. Otherwise, Y_{ij}^A is 0, i.e., activities i and j are processed simultaneously or j finishes some time before i begins. Because the ACP does not include precedence relationships, there are no prescribed values for the sequencing variables. The assignment variable Z_{ia}^A is equal to 1 if assessor a is assigned to activity i; otherwise $Z_{ia}^A = 0$. For the actors, the sequencing and assignment variables (Y_{ij}^P and Z_{ip}^P) are interpreted in the same way. Finally, variable V_{ca} is used to model the assessor-assignment rule, i.e., $V_{ca} = 1$ if assessor a is assigned to candidate c at least once.

The objective is to minimize the AC duration D.

$$\text{Min } D$$

The duration corresponds to the latest completion time of an activity that is defined by constraints (2.1).

$$D \geq S_i + p_i \qquad (i \in I) \tag{2.1}$$

Constraints (2.2)–(2.5) determine the resource-feasible start times of the activities. Constraints (2.2) are binding if candidate c completes activity i before the start of activity j. Otherwise, constraints (2.3) are binding. Because candidate c is not required during the evaluation time, activity j can start at most p_i^A time units before the completion of activity i (cf. Figure 2.3).

$$S_j \geq S_i - M + (p_i - p_i^A + M)Y_{ij}^C \qquad (c \in C, i, j \in I_c : i < j) \tag{2.2}$$

$$S_i \geq S_j - M + (p_j - p_j^A + M)(1 - Y_{ij}^C) \qquad (c \in C, i, j \in I_c : i < j) \tag{2.3}$$

Constraints (2.4) and (2.5) enforce a sequence of activities for the assessors and actors, respectively. In the case that activity i is executed before activity j by the assessors, constraints (2.4) are binding. Because the assessors are not required during the preparation time, activity j can start at most p_j^C time units before the completion of activity i (cf. Figure 2.4). Similarly, constraints (2.5) are binding if activity i is executed before activity j by the actors. For the actors, activity i is completed after $p_i - p_i^A + p_i^P$ time units.

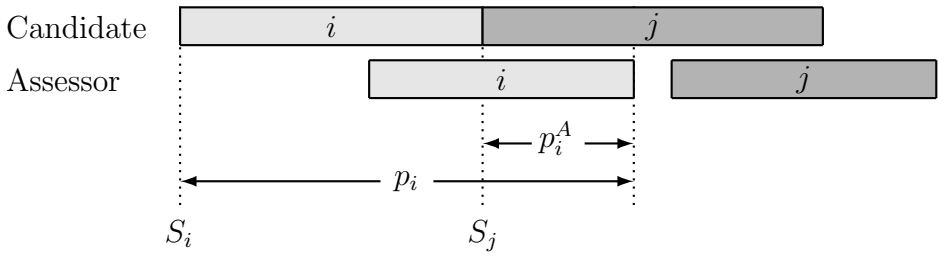


Figure 2.3: Minimum time lag between start times of activities i and j for candidates

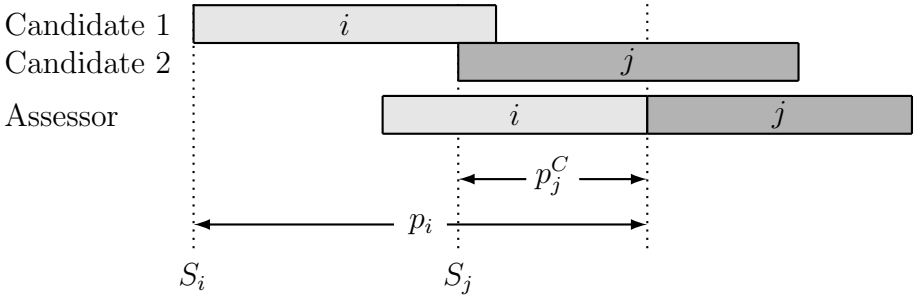


Figure 2.4: Minimum time lag between start times of activities i and j for assessors

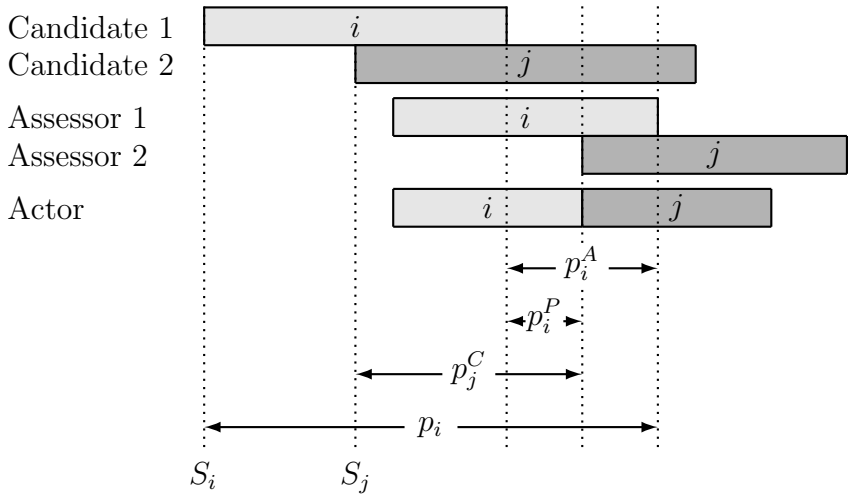


Figure 2.5: Minimum time lag between start times of activities i and j for actors

Activity j can start at most p_j^C time units before that completion time (cf. Figure 2.5).

$$S_j \geq S_i - M + (p_i - p_j^C + M)Y_{ij}^A \qquad (i,j \in I^A : i \neq j) \tag{2.4}$$
$$S_j \geq S_i - M + (p_i - p_i^A + p_i^P - p_j^C + M)Y_{ij}^P \qquad (i,j \in I^P : i \neq j) \tag{2.5}$$

Constraints (2.6) ensure that the lunch breaks are scheduled within the prescribed time window.

$$ES^L \leq S_i \leq LS^L \qquad (i \in I^L) \tag{2.6}$$

Constraints (2.7) and (2.8) imply that the required numbers of assessors and actors are assigned to each activity.

$$\sum_{a \in A} Z_{ia}^A = r_i^A \qquad (i \in I^A) \tag{2.7}$$
$$\sum_{p \in P} Z_{ip}^P = r_i^P \qquad (i \in I^P) \tag{2.8}$$

Constraints (2.9) and (2.10) link the assignment variables to the sequencing variables. If the same assessor a or the same actor p is assigned to two activities i and j, then a sequence between these two activities is enforced.

$$Y_{ij}^A + Y_{ji}^A \geq Z_{ia}^A + Z_{ja}^A - 1 \qquad (i,j \in I^A, a \in A : i < j) \tag{2.9}$$
$$Y_{ij}^P + Y_{ji}^P \geq Z_{ip}^P + Z_{jp}^P - 1 \qquad (i,j \in I^P, p \in P : i < j) \tag{2.10}$$

Constraints (2.11) and (2.12) ensure that either activity i precedes activity j, j precedes i, or i and j are processed in parallel.

$$Y_{ij}^A + Y_{ji}^A \leq 1 \qquad (i,j \in I^A : i < j) \tag{2.11}$$
$$Y_{ij}^P + Y_{ji}^P \leq 1 \qquad (i,j \in I^P : i < j) \tag{2.12}$$

Constraints (2.13) enforce that the number of assessors assigned to each candidate lies within the bounds imposed by the assessor-assignment rule.

$$\left\lfloor \frac{|A|}{2} \right\rfloor \leq \sum_{a \in A} V_{ca} \leq \left\lceil \frac{|A|}{2} \right\rceil + 1 \qquad (c \in C) \tag{2.13}$$

Constraints (2.14) determine whether an assessor a has been assigned to a candidate c at

least once. V_{ca} must be equal to 1 if assessor a is assigned to at least one activity that requires candidate c. If assessor a is never assigned to an activity that requires candidate c, then V_{ca} must be equal to 0.

$$\sum_{i \in I_c \setminus I^L} \frac{Z_{ia}^A}{|I_c \setminus I^L|} \leq V_{ca} \leq \sum_{i \in I_c \setminus I^L} Z_{ia}^A \qquad (c \in C, a \in A) \qquad (2.14)$$

Finally, constraints (2.15) model the no-go relationships.

$$V_{ca} = 0 \qquad ((c,a) \in N) \qquad (2.15)$$

In sum, formulation (CT–A) reads as follows:

$$\text{(CT–A)} \quad \begin{cases} \text{Min } D & \\ \text{s.t. } (2.1)\text{–}(2.15) & \\ \quad S_i \geq 0 & (i \in I) \\ \quad Y_{ij}^C \in \{0,1\} & (c \in C,\ i,j \in I_c :\ i < j) \\ \quad Y_{ij}^A \in \{0,1\} & (i,j \in I^A :\ i \neq j) \\ \quad Y_{ij}^P \in \{0,1\} & (i,j \in I^P :\ i \neq j) \\ \quad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\ \quad Z_{ia}^A \in \{0,1\} & (i \in I^A,\ a \in A) \\ \quad Z_{ip}^P \in \{0,1\} & (i \in I^P,\ p \in P) \end{cases}$$

2.4.2 Formulation CT–F

In this section, we present the continuous-time formulation with resource-flow variables (CT–F), which is based on the RCPSP formulation of Artigues et al. (2003). This MIP formulation was first proposed in Zimmermann and Trautmann (2014). The following explanations closely follow that study.

To model the resource flows, formulation CT–F requires the dummy activities 0 and $n+1$; both have a duration of zero, and $r_0^A = r_{n+1}^A = |A|$ ($r_0^P = r_{n+1}^P = |P|$) is equal to the total number of available assessors (actors). Variable F_{ij}^C (F_{ij}^A, F_{ij}^P) denotes the quantity of candidates (assessors, actors) sent from activity i (upon completion) to activity j (at the beginning). This resource flow prevents the corresponding activities from being executed simultaneously. For the assessors (actors), the sequencing variable Y_{ij}^A (Y_{ij}^P) is equal to 1 if some assessors (actors) are sent from activity i to activity j. Because each activity

requires exactly one candidate, any flow of candidates between two activities will be either 0 or 1. Since the resource-flow variable F_{ij}^C is defined as binary, this variable is used simultaneously as a resource-flow and as a sequencing variable. As a sequencing variable, F_{ij}^C equals 1 if and only if activity j is executed after activity i.

The following constraints have to be considered. Constraints (2.16) determine resource-feasible start times of the activities for the candidates. The feasible start times of the activities for the assessors and actors are determined as in formulation CT–A. Constraints (2.16) are binding if a candidate is sent from activity i to activity j ($F_{ij}^C = 1$).

$$S_j \geq S_i - M + (p_i - p_i^A + M)F_{ij}^C \qquad (c \in C;\ i,j \in I_c :\ i \neq j) \tag{2.16}$$

Constraints (2.17)–(2.22) are the resource-flow conservation constraints. Constraints (2.17) ensure that each activity i sends 1 unit of resource $c \in C$ to either an activity $j \neq i$ or the dummy activity $n+1$ (if activity i is the last activity performed by candidate c). Constraints (2.18) ensure that each activity j receives 1 unit of resource $c \in C$ from either an activity $i \neq j$ or the dummy activity 0 (if activity j is the first activity performed by candidate c).

$$\sum_{j \in I_c \cup \{n+1\}:\ j \neq i} F_{ij}^C = 1 \qquad (c \in C;\ i \in I_c \cup \{0\}) \tag{2.17}$$

$$\sum_{i \in I_c \cup \{0\}:\ i \neq j} F_{ij}^C = 1 \qquad (c \in C;\ j \in I_c \cup \{n+1\}) \tag{2.18}$$

Constraints (2.19)–(2.22) conserve the resource flow of assessors and actors, respectively. The number of assessors r_i^A (actors r_i^P) required by activity i must be sent to and received from other activities that require the same resource.

$$\sum_{j \in I^A \cup \{n+1\}:\ j \neq i} F_{ij}^A = r_i^A \qquad (i \in I^A \cup \{0\}) \tag{2.19}$$

$$\sum_{j \in I^P \cup \{n+1\}:\ j \neq i} F_{ij}^P = r_i^P \qquad (i \in I^P \cup \{0\}) \tag{2.20}$$

$$\sum_{i \in I^A \cup \{0\}:\ i \neq j} F_{ij}^A = r_j^A \qquad (j \in I^A \cup \{n+1\}) \tag{2.21}$$

$$\sum_{i \in I^P \cup \{0\}:\ i \neq j} F_{ij}^P = r_j^P \qquad (j \in I^P \cup \{n+1\}) \tag{2.22}$$

Constraints (2.23) and (2.24) link the resource-flow variables to the sequencing variables

for assessors and actors, respectively.

$$F_{ij}^A \leq \min(r_i^A, r_j^A) Y_{ij}^A \qquad (i,j \in I^A :\ i \neq j) \tag{2.23}$$
$$F_{ij}^P \leq \min(r_i^P, r_j^P) Y_{ij}^P \qquad (i,j \in I^P :\ i \neq j) \tag{2.24}$$

The sequencing variables Y_{ij}^A and Y_{ij}^P are only used to link the flow variables F_{ij}^A and F_{ij}^P to the start times of the activities. The flow variables F_{ij}^A and F_{ij}^P can be greater than 1. For this reason, they cannot be used as sequencing variables.

Constraints (2.1), which determine the AC duration D, and the sequencing constraints for the assessors (2.4) and actors (2.5), and constraints (2.6), which specify the time window for the lunch breaks, are also included. The same applies to the assessor-assignment constraints (2.7), (2.9), and (2.11)–(2.15).

In sum, formulation (CT–F) reads as follows:

$$\text{(CT–F)} \quad \begin{cases} \text{Min } D & \\ \text{s.t. } (2.16)\text{–}(2.24) & \\ \quad (2.1), (2.4)\text{–}(2.7), (2.9) & \\ \quad (2.11)\text{–}(2.15) & \\ \quad S_i \geq 0 & (i \in I) \\ \quad F_{ij}^C \in \{0,1\} & (c \in C;\ i,j \in I_c \cup \{0, n+1\} :\ i \neq j) \\ \quad F_{ij}^A \geq 0 & (i,j \in I^A \cup \{0, n+1\} :\ i \neq j) \\ \quad F_{ij}^P \geq 0 & (i,j \in I^P \cup \{0, n+1\} :\ i \neq j) \\ \quad Y_{ij}^A \in \{0,1\} & (i,j \in I^A :\ i \neq j) \\ \quad Y_{ij}^P \in \{0,1\} & (i,j \in I^P :\ i \neq j) \\ \quad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\ \quad Z_{ia}^A \in \{0,1\} & (i \in I^A,\ a \in A) \end{cases}$$

2.4.3 Formulation CT–O

In this section, we present the continuous-time formulation with overlapping variables (CT–O), which is based on the RCPSP formulation of Kopanos et al. (2014).

For activities that cannot be processed in parallel (i.e., two activities which require the same candidate), we use the sequencing variables Y_{ij}^C. For activities that can be processed in parallel, the resource constraints are modeled with the following binary variables.

- For the assessors and the actors, we introduce the sequencing variables $\widehat{Y}_{ij}$. Specif-

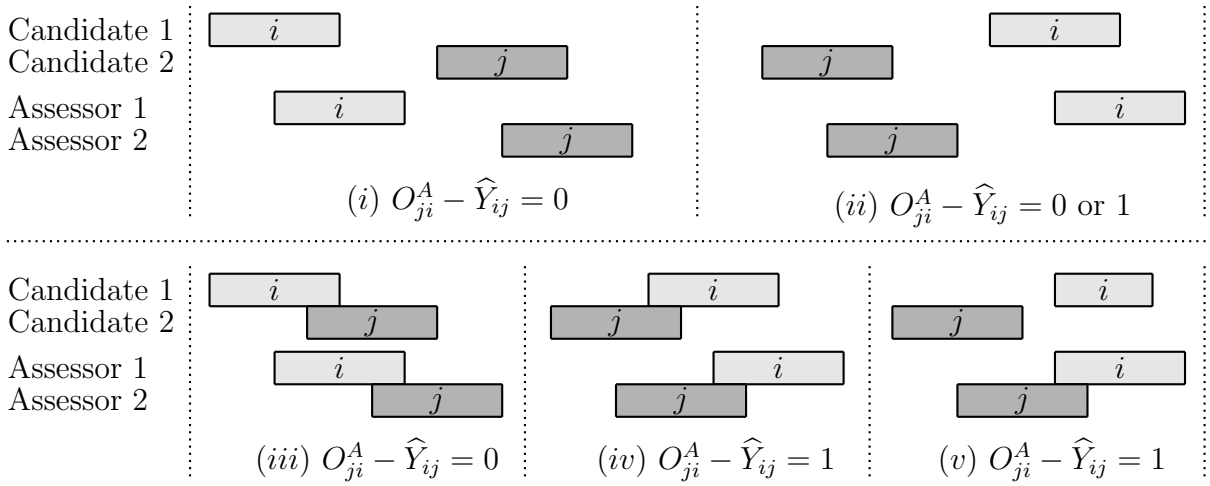


Figure 2.6: Five possible cases (i)–(v) that illustrate the values of the sequencing and overlapping variables

ically, $\widehat{Y}_{ij} = 1$ if activity i starts before or at the same time as activity j for the assessors. These sequencing variables are not defined separately for assessors and actors, because the activities start at the same time for them.

- For the assessors, we introduce the overlapping variables O^A_{ji}. Specifically, $O^A_{ji} = 1$ if activity j finishes after the start of activity i for the assessors. If activity j finishes before or at the same time as activity i starts, then O^A_{ji} is equal to 0 or 1. The overlapping variables for the actors O^P_{ji} are defined in the same way.

To illustrate how these variables jointly determine whether two activities $i, j \in I^A$ are processed in parallel by the assessors, several possible cases are depicted in Figure 2.6. For case (ii), the variable O^A_{ji} can be equal to zero or one, but for cases (iv) and (v), the variable must be equal to one.

Constraints (2.25) determine the resource-feasible start times of the activities for the candidates. Constraints (2.26) ensure that either activity i precedes activity j, or j precedes i. In contrast to constraints (2.2) and (2.3), the sequencing variables Y^C_{ij} are used for any pair of activities involving the same candidate.

$$S_i + p_i - p^A_i \leq S_j + MY^C_{ji} \qquad (c \in C, i, j \in I_c : i \neq j) \qquad (2.25)$$

$$Y^C_{ij} + Y^C_{ji} = 1 \qquad (c \in C, i, j \in I_c : i > j) \qquad (2.26)$$

Constraints (2.27)–(2.29) determine the resource-feasible start times of the activities which can be processed in parallel. Thereby, parameter λ is used to exclude some symmetric solutions, i.e., for two activities $i > j$ which start at the same time, it is specified that

$\widehat{Y}_{ji} = 1$ and $\widehat{Y}_{ij} = 0$. As proposed in Kopanos et al. (2014), we set $\lambda = 0.1$.

$$S_j + p_j^C \leq S_i + p_i^C + M\widehat{Y}_{ij} \qquad (i, j \in I^A : i > j) \tag{2.27}$$
$$S_i + p_i^C + \lambda \leq S_j + p_j^C + (M + \lambda)\widehat{Y}_{ji} \qquad (i, j \in I^A : i > j) \tag{2.28}$$
$$\widehat{Y}_{ij} + \widehat{Y}_{ji} = 1 \qquad (i, j \in I^A : i > j) \tag{2.29}$$

Constraints (2.30) and (2.31) link the overlapping variables to the start times of the activities.

$$(S_j + p_j) - (S_i + p_i^C) \leq MO_{ji}^A \qquad (i, j \in I^A : i \neq j) \tag{2.30}$$
$$(S_j + p_j - p_j^A + p_j^P) - (S_i + p_i^C) \leq MO_{ji}^P \qquad (i, j \in I^P : i \neq j) \tag{2.31}$$

Constraints (2.32) and (2.33) ensure that all activities that are executed in parallel do not require more than the available number of assessors and actors, respectively. Thereby, the term $O_{ji}^A - \widehat{Y}_{ij} = 1$ if activity j starts before activity i and if both activities overlap for the assessors. The same applies to the actors.

$$r_i^A + \sum_{j \in I^A : j \neq i} r_j^A (O_{ji}^A - \widehat{Y}_{ij}) \leq |A| \qquad (i \in I^A) \tag{2.32}$$
$$r_i^P + \sum_{j \in I^P : j \neq i} r_j^P (O_{ji}^P - \widehat{Y}_{ij}) \leq |P| \qquad (i \in I^P) \tag{2.33}$$

Constraints (2.34) and (2.35) ensure that the terms $O_{ji}^A - \widehat{Y}_{ij}$ and $O_{ji}^P - \widehat{Y}_{ij}$ are greater than or equal to zero.

$$\widehat{Y}_{ij} \leq O_{ji}^A \qquad (i, j \in I^A : i \neq j) \tag{2.34}$$
$$\widehat{Y}_{ij} \leq O_{ji}^P \qquad (i, j \in I^P : i \neq j) \tag{2.35}$$

Constraints (2.36) link the sequencing and overlapping variables to the assignment variables. If the same assessor a is assigned to two activities i and j, then both activities cannot overlap for the assessors.

$$(O_{ji}^A - \widehat{Y}_{ij}) + Z_{ia}^A + Z_{ja}^A \leq 2 \qquad (a \in A,\ i, j \in I^A : i \neq j) \tag{2.36}$$

Constraints (2.1), which determine the AC duration D, and constraints (2.6), which specify the time window for the lunch breaks, are also included. The same applies to the assessor-assignment constraints (2.13)–(2.15).

In sum, formulation (CT–O) reads as follows:

$$
\text{(CT–O)} \quad \begin{cases}
\text{Min } D & \\
\text{s.t. } (2.25)\text{–}(2.36) & \\
\quad (2.1), (2.6), (2.7), (2.13)\text{–}(2.15) & \\
\quad S_i \geq 0 & (i \in I) \\
\quad Y_{ij}^C \in \{0,1\} & (c \in C,\ i,j \in I_c : i \neq j) \\
\quad \widehat{Y}_{ij} \in \{0,1\} & (i,j \in I^A : i \neq j) \\
\quad O_{ji}^A \in \{0,1\} & (i,j \in I^A : i \neq j) \\
\quad O_{ji}^P \in \{0,1\} & (i,j \in I^P : i \neq j) \\
\quad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\
\quad Z_{ia}^A \in \{0,1\} & (i \in I^A,\ a \in A)
\end{cases}
$$

2.4.4 Formulation DT–P

In this section, we present the discrete-time formulation with pulse variables (DT–P), which is based on the RCPSP formulation of Pritsker et al. (1969). This formulation involves the discretization of the planning horizon into uniform time intervals. The endpoints of a time interval are denoted by the time points t and $t+1$, respectively $(t = 0, \ldots, T-1)$. Binary pulse variables X_{it} state if activity i starts at time t. For each time point t, resource constraints are formulated that ensure that the resource capacities are not violated. We extend the resource constraints of the RCPSP formulation such that the preparation and evaluation times of the AC activities are considered.

For the ACP, the following constraints have to be taken into consideration. The AC duration corresponds to the latest completion time of an activity, which is defined by constraints (2.37).

$$
D \geq \sum_{t=0}^{T-p_i} (t + p_i) X_{it} \qquad (i \in I) \tag{2.37}
$$

Constraints (2.38) and (2.39) ensure that each activity starts once. Furthermore, constraints (2.39) state that the lunch breaks are scheduled within the prescribed time win-

dow.

$$\sum_{t=0}^{T-p_i} X_{it} = 1 \qquad (i \in I \backslash I^L) \tag{2.38}$$

$$\sum_{t=ES^L}^{LS^L} X_{it} = 1 \qquad (i \in I^L) \tag{2.39}$$

Constraints (2.40) to (2.42) ensure that the resource capacities are not violated. Constraints (2.40) ensure that each candidate performs at most one activity at the same time t. Candidate c performs activity i at time t if the activity started between time $t-(p_i-p_i^A)+1$ and t. Constraints (2.41) and (2.42) ensure that all activities that are scheduled in parallel do not require more than the maximum available numbers of assessors and actors, respectively. An assessor performs activity i at time t if the activity started between time $t-p_i+1$ and $t-p_i^C$. An actor performs activity i at time t if the activity started between time $t-(p_i-p_i^A+p_i^P)+1$ and $t-p_i^C$.

$$\sum_{i \in I_c} \sum_{\tau=\max(0,t-p_i+p_i^A+1)}^{t} X_{i\tau} \leq 1 \qquad (c \in C, t = 0, \ldots, T) \tag{2.40}$$

$$\sum_{i \in I^A} \sum_{\tau=\max(0,t-p_i+1)}^{t-p_i^C} r_i^A X_{i\tau} \leq |A| \qquad (t = 0, \ldots, T) \tag{2.41}$$

$$\sum_{i \in I^P} \sum_{\tau=\max(0,t-p_i+p_i^A-p_i^P+1)}^{t-p_i^C} r_i^P X_{i\tau} \leq |P| \qquad (t = 0, \ldots, T) \tag{2.42}$$

Additionally, the assessor-assignment constraints (2.7), (2.9), (2.11), and (2.13)–(2.15) are also included. Constraints (2.43) link the variables X_{it} to the sequencing variables Y_{ij}^A.

$$\sum_{t=0}^{T-p_j} tX_{jt} \geq \sum_{t=0}^{T-p_i} tX_{it} - M + (p_i - p_j^C + M)Y_{ij}^A \qquad (i, j \in I^A : \ i \neq j) \tag{2.43}$$

In sum, formulation (DT–P) reads as follows:

$$
\text{(DT–P)} \quad \begin{cases}
\text{Min } D & \\
\text{s.t. } (2.37)\text{–}(2.43) & \\
\quad (2.7), (2.9), (2.11), (2.13)\text{–}(2.15) & \\
\quad X_{it} \in \{0,1\} & (i \in I, t = 0, \ldots, T) \\
\quad Y_{ij}^{A} \in \{0,1\} & (i,j \in I^{A} :\ i \neq j) \\
\quad V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\
\quad Z_{ia}^{A} \in \{0,1\} & (i \in I^{A},\ a \in A)
\end{cases}
$$

2.4.5 Formulation DT–O

In this section, we present the discrete-time formulation with on/off variables (DT–O), which is based on the RCPSP formulation of Kopanos et al. (2014). For the RCPSP, Kopanos et al. (2014) extend the formulation of Pritsker et al. (1969) with binary on/off variables W_{it}, which specify if activity i is in progress at time t. With these variables, the resource constraints can be modeled in a different manner than in Pritsker et al. (1969).

For the ACP, we extend the formulation DT–P (cf. Section 2.4.4) with binary on/off variables. Due to the preparation and the evaluation time, these on/off variables must be defined individually for candidates, assessors, and actors. However, this results in a large number of additional variables, which has a negative impact on the performance. For this reason, we only define the on/off variables for the candidates, and take the resource constraints of DT–P for the assessors and the actors. Hence, the resource constraints (2.40) for the candidates are replaced by constraints (2.44)–(2.46).

Constraints (2.44) ensure that each candidate performs at most one activity at a time.

$$
\sum_{i \in I_c : t \leq T - p_i^A - 1} W_{it} \leq 1 \qquad (c \in C, t = 0, \ldots, T) \tag{2.44}
$$

Constraints (2.45) link the pulse variables X_{it} to the on/off variables W_{it}.

$$
W_{it} = \sum_{\tau = \max(0, t - p_i + p_i^A + 1)}^{t} X_{i\tau} \qquad (i \in I, t = 0, \ldots, T - p_i^A - 1) \tag{2.45}
$$

Constraints (2.46) are valid equalities that tighten the formulation.

$$\sum_{t=0}^{T-p_i^A-1} W_{it} = p_i - p_i^A \qquad (i \in I) \tag{2.46}$$

In sum, formulation (DT–O) reads as follows:

$$\text{(DT–O)} \quad \begin{cases} \text{Min } D & \\ \text{s.t. } (2.44)\text{–}(2.46) & \\ \quad (2.37)\text{–}(2.39), (2.41)\text{–}(2.43) & \\ \quad (2.7), (2.9), (2.11), (2.13)\text{–}(2.15) & \\ \quad X_{it} \in \{0,1\} & (i \in I, t = 0, \ldots, T) \\ \quad W_{it} \in \{0,1\} & (i \in I, t = 0, \ldots, T - p_i^A - 1) \\ \quad Y_{ij}^A \in \{0,1\} & (i, j \in I^A : \ i \neq j) \\ \quad V_{ca} \in \{0,1\} & (c \in C, \ a \in A) \\ \quad Z_{ia}^A \in \{0,1\} & (i \in I^A, \ a \in A) \end{cases}$$

2.5 Lower bounds

In this section, we derive some lower bounds for the AC duration. In Section 2.5.1, we present four lower bounds based on the assessors' workload. In Section 2.5.2, we present two lower bounds based on the candidates' workload.

2.5.1 Lower bounds based on the assessors' workload

In this section, we present four different lower bounds ($LB_1, \ldots, LB_4$) that are based on the assessors' workload. In contrast to lower bounds LB_1 and LB_2, lower bounds LB_3 and LB_4 consider the no-go relationships.

Lower bound LB_1 corresponds to the average workload of the assessors increased by the shortest preparation time of an activity. This preparation time is included because the assessors are never required before that time. The lower bound LB_1 reads as follows.

$$LB_1 = \left\lceil \sum_{i \in I^A} \frac{r_i^A (p_i - p_i^C)}{|A|} \right\rceil + \min_{i \in I^A} p_i^C$$

Lower bound LB_2 is obtained by considering only the activities that require two assessors. The total workload of these activities must be completed by an even number of

assessors. Hence, if the number of assessors $|A|$ is odd, then the following lower bound LB_2 is valid.

$$LB_2 = \left\lceil \sum_{i \in I^A : r_i^A = 2} \frac{2(p_i - p_i^C)}{|A| - 1} \right\rceil + \min_{i \in I^A} p_i^C$$

Lower bound LB_3 takes the no-go relationships of each assessor into consideration. The workload of all activities to which assessor a cannot be assigned due to no-go relationships is evenly distributed among the remaining $|A| - 1$ assessors and increased by the shortest preparation time. For each assessor $a \in A$, this corresponds to a lower bound.

$$LB_3 = \max_{a \in A} \left\lceil \sum_{c \in C : (c,a) \in N} \sum_{i \in I_c} \frac{r_i^A (p_i - p_i^C)}{|A| - 1} \right\rceil + \min_{i \in I^A} p_i^C$$

Lower bound LB_4 combines the underlying ideas of LB_2 and LB_3. We only consider activities that require two assessors and for which the corresponding candidates have a no-go relationship with assessor a. For these activities, an even number of assessors is required at any time. However, if the number of assessors is even and assessor a cannot be assigned to these activities due to the no-go relationships, it follows that one assessor $a^* \neq a$ is not needed. Hence, the workload of all activities that require two assessors and to which assessor a cannot be assigned is evenly distributed among the remaining $|A| - 2$ assessors. Again, the shortest preparation time of an activity is added to increase the lower bound. Hence, if the number of assessors $|A|$ is even, then lower bound LB_4 is valid.

$$LB_4 = \max_{a \in A} \left\lceil \sum_{c \in C : (c,a) \in N} \sum_{i \in I_c : r_i^A = 2} \frac{2(p_i - p_i^C)}{|A| - 2} \right\rceil + \min_{i \in I^A} p_i^C$$

2.5.2 Lower bounds based on the candidates' workload

In this section, we present two lower bounds for the AC duration based on the candidates' workload. The first lower bound (LB_5) is valid in general, and the second lower bound (LB_6) is only valid under certain conditions. Because each candidate must perform the same tasks, we do not need to differentiate between different candidates. Hence, in the following, we consider the tasks to be executed by each candidate and the lunch break rather than activities for individual candidates. The set of tasks and the lunch break are denoted by Q and l, respectively. It should be noted that the lunch break is not included in Q. Let p_q, p_q^C, and p_q^A be the duration, the preparation time, and the assessors' evaluation time of task $q \in Q$, respectively. The duration of the lunch break is p_l, and its preparation

time (p_l^C) and evaluation time (p_l^A) are zero.

Because the tasks and the lunch break must be performed sequentially, lower bound LB_5 is valid.

$$LB_5 = \sum_{q \in Q \cup \{l\}} \left(p_q - p_q^A\right)$$

The term $\min_{q \in Q \cup \{l\}} p_q^A$ could be added to LB_5 because the AC cannot end before all tasks and the lunch break are completed. However, the evaluation time of the lunch break is always equal to zero and, thus, this term is always zero. The lunch break cannot be excluded from this term, because each candidate can have the lunch break at the end if the latest possible start time is not violated.

To motivate lower bound LB_6, we first consider an illustrative example with two candidates and three assessors. Each candidate has to perform a task (activities k_1 and k_2) that requires two assessors and a lunch break (activities l_1 and l_2); activities k_1 and k_2 cannot be scheduled in parallel due to the limited number of assessors. Figure 2.7 depicts two feasible schedules for this example. In the schedule on the left, both candidates have the lunch break at the end. Due to the limited number of assessors, candidate C_2 has a waiting time. In this case, the AC duration D corresponds to the lower bound LB_5 plus the waiting time. In the schedule on the right, candidate C_2 performs the lunch break first. In this case, the AC duration D correspond to the lower bound LB_5 plus the evaluation time of the task.

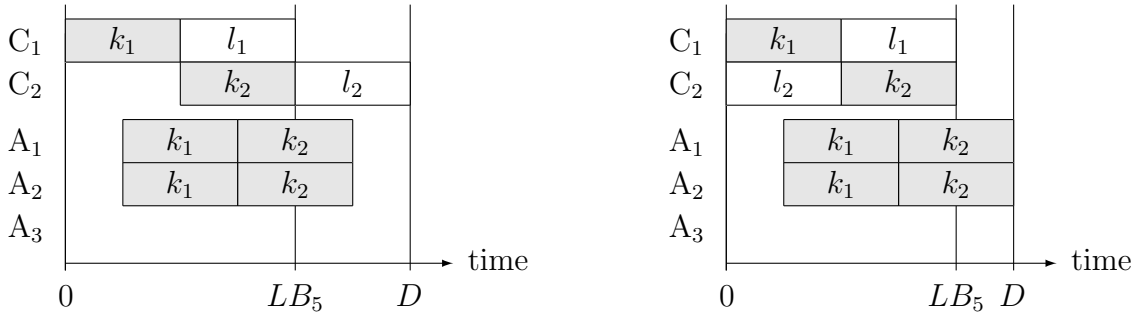


Figure 2.7: Schedules of an example with (left) and without (right) waiting time for the candidates

In this example, either a candidate has a waiting time, or the last activity of a candidate does not correspond to the lunch break. With this in mind, we propose lower bound LB_6, which is valid under certain conditions. According to our industry partner, these conditions are fulfilled by a considerable number of real-life instances.

Theorem 1. *Let r be a task with the shortest evaluation time. If (i) $\lfloor |A|/2 \rfloor < |C|$ and (ii) all tasks except task r require two or more assessors, then the following lower bound*

is valid.

$$LB_6 = \delta_0 + \min(\delta_1, \delta_2)$$

whereas:

$$\delta_0 = \sum_{q \in Q \cup \{l\}} \left(p_q - p_q^A\right)$$

$$\delta_1 = \min_{q \in Q \setminus \{r\}} p_q^A$$

$$\delta_2 = \min_{q \in Q \setminus \{r\}} (p_q - p_q^C - p_q^A) + \min_{q \in Q \setminus \{r\}} p_q^A - \max(p_l, p_r - p_r^A)$$

Proof. If the conditions (i) and (ii) hold for a given problem instance, any feasible solution belongs either to case 1 or to case 2.

- Case 1: The last activity of at least one candidate does not correspond to a lunch break or an activity of task r. It results that after the candidate completes this last activity, the assessors have an evaluation time of at least δ_1. Hence, $\delta_0 + \delta_1$ is a lower bound if the solution belongs to case 1.

- Case 2: The last activity of each candidate either corresponds to a lunch break or an activity of task r. We show that in this case, at least one candidate has a waiting time of at least δ_2 because condition (i) implies that not all candidates can perform an activity that requires two assessors at the same time. δ_2 corresponds to the length of the minimum time interval during which the required number of assessors exceeds the number of available assessors.

  Let k denote an arbitrary task that requires two assessors. To determine δ_2, we first consider the four possibilities for ordering the last activities such that the lunch break or task r are performed at the end by each candidate (cf. Figure 2.8).

  a) The lunch break is performed at the end and preceded by task r. Task r is preceded by task k.

  b) Task r is performed at the end and preceded by the lunch break. The lunch break is preceded by task k.

  c) Task r is performed at the end and preceded by task k. The lunch break ends some time before task k.

  d) The lunch break is performed at the end and preceded by task k. Task r ends some time before task k.

  In Figure 2.8, the time point t_4 in a) and b) corresponds to the earliest possible finish time of task k for the assessors. The time points t_1, t_2, and t_3 correspond

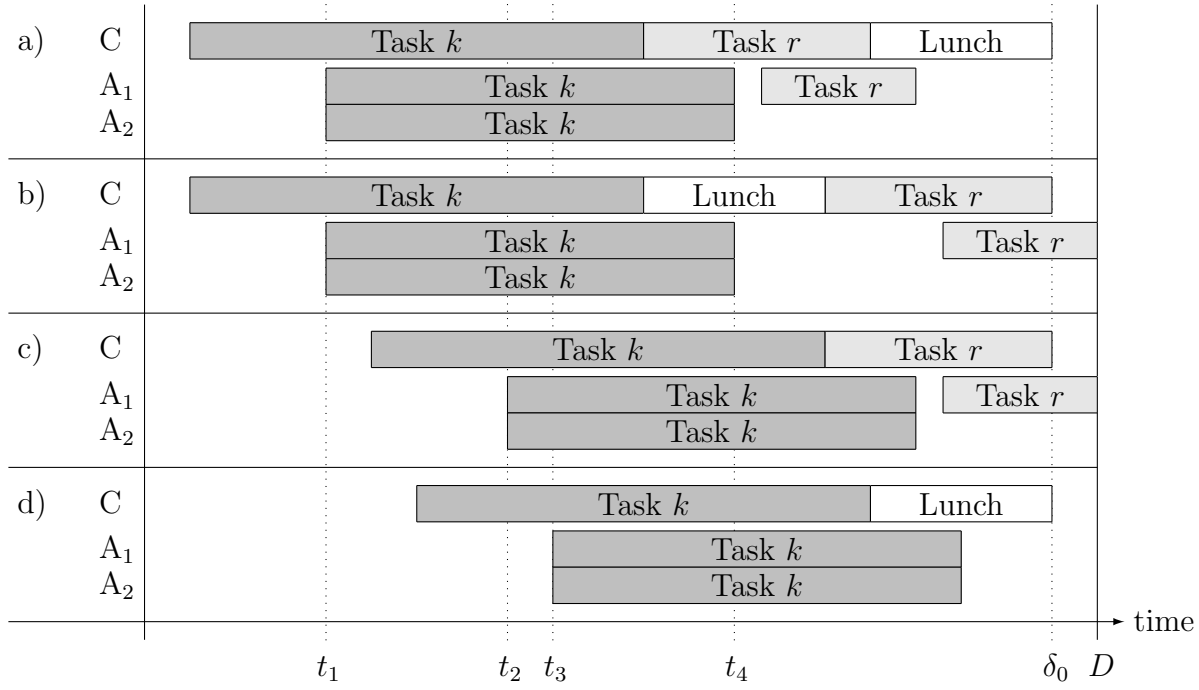


Figure 2.8: All possible orders of the last activities and corresponding assessor requirements

to the possible start times of task k for the assessors if no candidate has a waiting time. The values of these time points are as follows.

$$
\begin{aligned}
t_1 &= \delta_0 - p_l - (p_r - p_r^A) - (p_k - p_k^C - p_k^A) \\
t_2 &= \delta_0 - (p_r - p_r^A) - (p_k - p_k^C - p_k^A) \\
t_3 &= \delta_0 - p_l - (p_k - p_k^C - p_k^A) \\
t_4 &= \delta_0 - p_l - (p_r - p_r^A) + p_k^A
\end{aligned}
$$

Overall, the latest possible start time of task k for the assessors corresponds to

$$\max(t_1, t_2, t_3) = \delta_0 - \min(p_l, p_r - p_r^A) - (p_k - p_k^C - p_k^A).$$

If $t_4 > \max(t_1, t_2, t_3)$ and no candidate has a waiting time, then there is a time interval with a minimum length of $t_4 - \max(t_1, t_2, t_3)$ during which every candidate performs a task that requires two assessors. Because $\lfloor |A|/2 \rfloor < |C|$, the required number of assessors exceeds the available number of assessors in this interval. To resolve this conflict, at least one task k must be delayed, which leads to a minimum waiting time for at least one candidate of $t_4 - \max(t_1, t_2, t_3)$.

To derive a lower bound for the AC duration, we determine the smallest possible value of t_4 and the largest possible value of $\max(t_1, t_2, t_3)$ as follows.

$$t_4 \geq \delta_0 - p_l - (p_r - p_r^A) + \min_{q \in Q \setminus \{r\}} p_q^A$$
$$\max(t_1, t_2, t_3) \leq \delta_0 - \min(p_l, p_r - p_r^A) - \min_{q \in Q \setminus \{r\}} (p_q - p_q^C - p_q^A)$$

Hence, the minimum waiting time corresponds to

$$\begin{aligned} t_4 - \max(t_1, t_2, t_3) &\geq \min_{q \in Q \setminus \{r\}} (p_q - p_q^C - p_q^A) + \min_{q \in Q \setminus \{r\}} p_q^A - \max(p_l, p_r - p_r^A) \\ &= \delta_2. \end{aligned}$$

Thereby, we used $\alpha + \beta - \min(\alpha, \beta) = \max(\alpha, \beta)$, where α, β are two arbitrary numbers. Hence, $\delta_0 + \delta_2$ is a lower bound if the solution belongs to case 2.

Overall, $LB_6 = \delta_0 + \min(\delta_1, \delta_2)$ is a lower bound for the AC duration if conditions (i) and (ii) hold. □

In the performance analysis, we use the maximum of these problem-specific lower bounds. If for an instance the necessary conditions for any of the lower bounds are not fulfilled, we set their respective value to 0.

$$LB^+ = \max(LB_1, LB_2, \ldots, LB_6)$$

2.6 Comparative analysis

We implemented the MIP formulations presented in Section 2.4 in AMPL, and we used the Gurobi Optimizer 6.0.5 as solver. All calculations were performed on an HP workstation with an Intel Xeon 2.67 GHz CPU and 24 GB RAM. The computational experiment was performed using four real-life instances and 240 test instances derived from real-life data. We limited the CPU time of the solver to 3,600 seconds for the real-life instances and to 600 seconds for the test instances. We used Gurobi with its default settings. Additionally, we applied Gurobi with the parameter MIPFocus set to 1. The parameter MIPFocus determines the MIP solution strategy of the solver. When this parameter is set to 1, Gurobi focuses on quickly generating good feasible solutions rather than increasing the lower bound. The default setting is 0, which aims to balance between finding good feasible solutions and proving optimality. For the DT formulations, the upper bound of

Table 2.5: Real-life instances

Instance	$\|C\|$	$\|A\|$	$\|P\|$	$\|E\|$	$\|I\|$	No-go relationships
RL1	7	10	2	5	42	no
RL2	11	11	3	5	66	no
RL3	9	11	3	5	54	yes
RL4	6	9	3	5	36	no

the AC duration was set to $T = 200$ for all instances; this value is prescribed by the human resource provider.

In Section 2.6.1, we describe the instances that we used in our computational study. In Section 2.6.2, we discuss our computational results for the real-life instances. In Section 2.6.3, we provide the results for the test instances. In Section 2.6.4, we compare our problem-specific lower bounds.

2.6.1 Instances

The number of candidates $|C|$, assessors $|A|$, actors $|P|$, tasks $|E|$ and activities $|I|$ of the four real-life instances are listed in Table 2.5. The last column indicates whether at least one no-go relationship exists. We denote the real-life instances with RL1, ..., RL4.

To test the different MIP formulations, we additionally devised a test set with 240 test instances based on real-life data. For the RCPSP, the well-known test instances of Kolisch and Sprecher (1997) were generated by systematically varying the complexity factors resource strength (RS), resource factor (RF), and network complexity (NC). These factors are only partially applicable to generate the ACP instances. The factor NC corresponds to the average number of precedence relationships per activity. Because there are no precedence relationships among the activities of the AC, we do not require such a factor. The factors RF and RS correspond to the average portion of the resources used by an activity and the scarcity of the resources, respectively. The factor RF can be interpreted as the average number of assessors required by an activity. To ensure that the instances are as close to reality as possible, we selected real-life tasks with given requirements for assessors and actors. Hence, we do not require a factor such as RF. The factor RS can be interpreted as the scarcity of the assessors. We use a similar factor to determine the number of available assessors. In total, we generated the 240 test instances by varying five complexity factors. Thereby, the employed experimental levels of each complexity factor were based on real-life data provided by the human resource

management service provider. The complexity factors are as follows.

The complexity factors n^C and n^E correspond to the number of candidates and tasks, respectively, and determine the number of activities of an instance. The tasks were randomly selected from a set of 15 real-life tasks. The experimental levels $n^C \in \{4, 5, \ldots, 10, 11\}$ and $n^E \in \{4, 5\}$ were used.

The complexity factor a^S corresponds to the average number of assignments per assessor. This factor is used to determine the number of assessors n^A of an instance. The number of assessors is equal to the nearest integer to $\sum_{i \in I^A} r_i^A / a^S$; thus, the numerator corresponds to the total number of assessor assignments. The experimental levels $a^S \in \{6.0, 8.5, 10.4\}$ correspond to the observed real-life minimum, average, and maximum.

The complexity factor a^N corresponds to the proportion of assessors who have one or more no-go relationships (no-go assessors). The number of no-go assessors is given by the nearest integer to $a^N n^A$. The no-go assessors were randomly selected from the set of all assessors. The experimental levels $a^N \in \{\frac{1}{6}, \frac{1}{3}\}$ were used.

The complexity factor a^R corresponds to the average number of no-go relationships per no-go assessor. The number of no-go relationships is equal to the product of a^R and the number of no-go assessors. The no-go relationships were randomly assigned to pairs of candidates and no-go assessors such that (1) each no-go assessor has at least one no-go relationship and (2) at least $\lfloor |A| / 2 \rfloor$ different assessors can be assigned to each candidate. The experimental levels $a^R \in \{2, 3\}$ were used.

Because the actors are paid for each role play in which they actually perform, they are not considered to be a critical resource. Hence, the number of actors was set to 3 for all instances, which corresponds to the observed real-life maximum.

For each combination of complexity factor levels, an instance was generated; this leads to $8 \cdot 2 \cdot 3 \cdot 2 \cdot 2 = 192$ test instances. Additionally, $8 \cdot 2 \cdot 3 = 48$ test instances without no-go relationships (i.e., $a^N = a^R = 0$) were generated.

2.6.2 Computational results: real-life instances

For the real-life instances RL1, ..., RL4, the results obtained by the solver using the MIP formulations CT–A, CT–F, CT–O, DT–P, and DT–O with MIPFocus set to 0 are reported in Table 2.6. We compare the objective function values (D) with the lower bounds obtained by the solver (LB) and the maximum value over all problem-specific lower bounds (LB^+). For each instance, the best objective function values obtained are highlighted in boldface. Using the default solver settings, the solver obtains on average the best objective function values with CT–O and the highest lower bounds with DT–P.

Table 2.6: Results for real-life instances with MIPFocus set to 0

Instance	CT–A		CT–F		CT–O		DT–P		DT–O		LB^+
	D	LB	D	LB	D	LB	D	LB	D	LB	
RL1	89	67	90	37	**88**	74	128	81	95	71	82
RL2	136	59	158	36	**132**	49	149	103	173	72	110
RL3	**106**	62	121	36	107	49	125	80	118	63	90
RL4	83	70	86	36	**82**	74	87	81	86	80	82

Table 2.7: Results for real-life instances with MIPFocus set to 1

Instance	CT–A		CT–F		CT–O		DT–P		DT–O		LB^+
	D	LB	D	LB	D	LB	D	LB	D	LB	
RL1	**86**	49	**86**	36	88	49	98	76	88	70	82
RL2	**124**	49	128	36	129	54	159	70	150	69	110
RL3	102	49	**100**	36	108	49	118	59	114	63	90
RL4	**82**	56	84	36	84	55	**82**	82	**82**	76	82

For all real-life instances, these lower bounds are smaller than or equal to the problem-specific lower bound. The problem-specific lower bound of instance RL4 corresponds to the objective function value obtained with CT–O, i.e., this solution is optimal.

Table 2.7 lists the results obtained by the solver with MIPFocus set to 1. For each instance, the best objective function values obtained are highlighted in boldface. Except for CT–O, the average AC duration is improved. However, on average, the lower bounds are worse. CT–A devises the best solutions for three instances, CT–F for two instances, and DT–P and DT–O for one instance. The smallest instance (RL4) is even solved to optimality using formulation DT–P. Both, CT–A and DT–O, also find a solution with an optimal objective function value, but they do not prove optimality within the prescribed CPU time.

2.6.3 Computational results: test instances

Based on the number of activities $|I|$, we divide the 240 test instances into small-sized (20–34 activities, 75 instances), medium-sized (35–49 activities, 90 instances), and large-sized

(50–66 activities, 75 instances) instances. For these three ranges of $|I|$, the average number of variables and constraints for the different formulations are presented in Figure 2.9. Regardless of the number of activities, DT–O has the highest number of variables. For small- and medium-sized instances, DT–O has also the highest number of constraints. However, with an increasing number of activities, the number of constraints increases less for the DT formulations than for the CT formulations. For the large-sized instances, CT–O has the highest number of constraints.

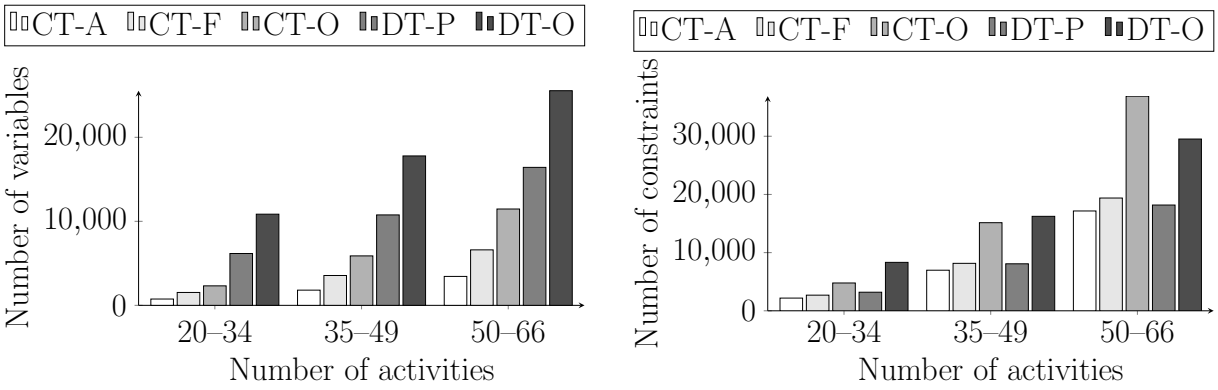


Figure 2.9: Average number of variables (left) and constraints (right)

Table 2.8 reports the average relative gaps between the obtained solutions and the problem-specific lower bound ($gap^+ = (D - LB^+)/D$), as well as the average relative gaps between the obtained solutions and the lower bounds obtained by the solver ($gap = (D - LB)/D$). To evaluate the quality of the solutions, we use gap^+. To evaluate the quality of the lower bounds provided by the solver, we use gap. For each solver setting used, the best results are highlighted in boldface.

Regardless of the solver settings employed, the best gap^+ is obtained with CT–A (10.3% for MIPFocus set to 0 and 9.3% for MIPFocus set to 1), and the worst gap^+ is obtained with DT–P. In contrast, the smallest gap is obtained with DT–O. Similarly to the results of Kopanos et al. (2014), better solutions are obtained with DT–O than with DT–P. We conclude that the CT formulations provide better solutions, and that the DT formulations provide better lower bounds. For all formulations, gap considerably exceeds gap^+. We deduce that the problem-specific lower bounds are considerably higher than the lower bounds obtained by the solver within the prescribed CPU time limit.

With CT–A, CT–O, and DT–P, feasible solutions are obtained for all 240 test instances within the prescribed CPU time limit. With CT–F and MIPFocus set to 0, feasible solutions are obtained only for 216 instances (i.e., 90% of the instances). With MIPFocus set to 1, this number increases to 235 (i.e., 97.9%); feasible solutions could not be obtained

Table 2.8: Aggregated results for all 240 test instances

Formulation	MIP-Focus	CT–A	CT–F	CT–O	DT–P	DT–O
Average gap^+ [in %]	0	**10.3**	15.1	12.5	27.7	19.4
	1	**9.3**	11.1	11.2	26.8	18.5
Average *gap* [in %]	0	44.7	59.8	50.4	37.5	**36.6**
	1	56.7	65.0	55.1	44.6	**37.8**
Number of feasible solutions	0	**240**	216	**240**	**240**	234
	1	**240**	235	**240**	**240**	238
Number of optimal solutions	0	**36**	29	32	22	19
	1	27	24	**30**	22	27
Number of best solutions	0	**170**	51	80	22	60
	1	**161**	69	81	27	57

for five of the large-sized instances.

To determine the number of optimal solutions, we compare the objective function value obtained with the maximum value over all problem-specific lower bounds and the lower bound obtained by the solver. With 36 instances, CT–A obtains the highest number of optimal solutions.

The number of best solutions corresponds to the number of times that a formulation generates a best solution. With MIPFocus set to 0, CT–A provides a best solution for 170 instances. This means that the other formulations generate better solutions for 70 instances only.

With MIPFocus set to 1, the average solution quality for all formulations is improved. This is indicated by a reduction of gap^+. For CT–F, this reduction is quite considerable (from 15.1% to 11.1%). This might indicate that the MIP solution strategy used by the solver exploits the resource-flow information in an efficient manner. However, the average *gap* is larger with MIPFocus set to 1 because this solver setting focuses less on improving the lower bounds but gives priority to the quick generation of good feasible solutions. Therefore, the number of feasible solutions is increased for CT–F. Surprisingly, for the CT formulations CT–A, CT–F and CT–O, the number of optimal solutions obtained is lower with MIPFocus set to 1.

Table 2.9 reports the average results for all instances with the same problem charac-

Table 2.9: Average gap^+ for different instance characteristics

Instance characteristics		MIP-Focus	Average gap^+				
			CT–A	CT–F	CT–O	DT–P	DT–O
$\|I\|$	20–34	0	**1.8**	3.1	2.6	10.7	11.0
		1	**2.4**	3.0	2.6	6.8	6.0
	35–49	0	**8.3**	14.9	11.4	28.0	12.7
		1	**7.8**	9.3	9.5	28.0	13.8
	50–66	0	**21.2**	31.8	23.6	44.4	36.5
		1	**18.1**	22.0	21.8	45.4	37.3
a^S	6	0	**10.1**	15.2	12.8	29.3	22.4
		1	**9.6**	11.3	11.9	23.4	20.3
	8	0	**12.1**	17.8	14.3	31.0	20.4
		1	**11.2**	12.9	12.5	32.2	19.4
	10.4	0	**8.8**	11.9	10.4	22.9	15.3
		1	**7.1**	9.0	9.1	24.8	15.8
a^N	0	0	**10.7**	16.9	12.5	25.9	18.8
		1	**9.2**	10.6	11.0	24.9	17.5
	0.17	0	**10.3**	14.9	12.8	27.4	19.4
		1	**9.4**	11.3	11.2	26.9	17.5
	0.33	0	**10.2**	14.5	12.2	29.0	19.6
		1	**9.3**	11.2	11.2	27.6	20.0
a^R	0	0	**10.7**	16.9	12.5	25.9	18.8
		1	**9.2**	10.6	11.0	24.9	17.5
	2	0	**10.3**	15.9	12.4	26.8	18.0
		1	**9.4**	11.3	10.8	27.0	17.8
	3	0	**10.2**	13.5	12.6	29.6	21.1
		1	**9.3**	11.2	11.7	27.6	19.8
f	11–13	0	**9.3**	13.0	12.2	25.6	10.6
		1	**8.6**	10.0	10.5	26.8	11.2
	13–15	0	**8.9**	13.5	10.7	24.4	18.8
		1	**7.3**	9.4	9.8	22.9	17.5
	15–17	0	**11.5**	17.0	13.5	30.3	23.2
		1	**10.7**	12.4	12.2	28.8	21.9

teristics. For each solver setting used, the best results are highlighted in boldface. The overall results show that with MIPFocus set to 1 the best solutions are obtained. However, for CT–A and small-sized instances, the solver performs better with MIPFocus set to 0.

The number of activities $|I|$ and the level of complexity factor a^S, which defines the number of available assessors, have a significant impact on both relative gaps. In contrast, the levels of complexity factors a^N and a^R, which define the no-go relationships, have no systematic impact on the relative gaps. Parameter f corresponds to the average duration of the activities. The performance of DT–O is affected most by the value of f. For instances with short activities ($11 \leq f \leq 13$), the performance of DT–O is almost as good as the performance of CT–A. However, for the instances with longer activities, the average gaps are much higher. Surprisingly, such an effect is not observed with DT–P.

According to the results obtained by Koné et al. (2011) for the RCPSP, DT formulations are better for instances with activities that have a short duration. Although the durations of the AC activities are quite short, we do not observe similar results for the ACP. Overall, the CT formulations provide the best solutions. A drawback of the DT formulations may be the large number of variables (cf. Figure 2.9) which depend on the number of time points considered. In the RCPSP, the number of variables is reduced considerably with a simple preprocessing like the definition of earliest and latest start times for the activities. However, this preprocessing is based on precedence relationships, which do not exist in the ACP. Considering the CT formulations, CT–A performs best, and CT–O performs better than CT–F.

2.6.4 Computational results: problem-specific lower bounds

Table 2.10 compares the six problem-specific lower bounds presented in Section 2.5. The last row shows the number of instances for which the different lower bounds obtained the highest values. LB_1 and LB_2 each provide the highest lower bounds for more than 90 instances. However, lower bounds that consider no-go relationships (LB_3 and LB_4) only provide the highest values for a few instances. If the conditions for LB_6 hold, this lower bound provides the highest values for 22 instances.

2.7 Conclusions

Comparisons of alternative MIP formulations in the literature for project scheduling problems are primarily based on generic test instances. In this study, we analyzed the performance of two discrete-time and three continuous-time MIP formulations in a real-life

Table 2.10: Comparison of problem-specific lower bounds

Lower bound	LB_1	LB_2	LB_3	LB_4	LB_5	LB_6
Number of instances with best lower bound	93	90	0	8	32	22

application of project scheduling. We considered the problem of planning assessment centers. For this problem, we developed new MIP formulations, and we provided problem-specific lower bounds. In contrast to the results generally obtained for the RCPSP, our comparative study indicates that the CT formulations outperform the DT formulations in terms of solution quality. However, using the DT formulations, the best MIP-based lower bounds are obtained.

The assessment center planning problem is an interesting and challenging optimization problem for future research. An important area is the development of heuristic solution procedures. Preliminary versions of an MIP-based heuristic and a list-scheduling heuristic are presented in Rihm and Trautmann (2016) and Zimmermann and Trautmann (2015). In the MIP-based heuristic, first, the activities are scheduled without assessor assignments; second, the assessors are assigned to the activities using the CT formulation with resource-flow variables presented in this study. In the list-scheduling heuristic, the activities are scheduled sequentially based on problem-specific priority rules. The MIP formulations and the problem-specific lower bounds presented in this paper can be used to analyze the performance of such heuristic approaches.

Bibliography

Alvarez-Valdes, R., Tamarit, J., 1993. The project scheduling polyhedron: dimension, facets and lifting theorems. Eur J Oper Res 67 (2), 204–220.

Ambrosino, D., Paolucci, M., Sciomachen, A., 2015. Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. Flex Serv Manuf J 27 (2–3), 263–284.

Artigues, C., Koné, O., Lopez, P., Mongeau, M., 2015. Mixed-integer linear programming formulations. In: Schwindt, C., Zimmermann, J. (Eds.), Handbook on Project Management and Scheduling Vol. 1. Springer, Cham, pp. 17–41.

Artigues, C., Michelon, P., Reusser, S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. Eur J Oper Res 149 (2), 249–267.

Bianco, L., Caramia, M., 2013. A new formulation for the project scheduling problem under limited resources. Flex Serv Manuf J 25 (1–2), 6–24.

Bixby, R. E., 2012. A brief history of linear and mixed-integer programming computation. Doc Math Extra Volume ISMP, 107–121.

Chen, X., Grossmann, I., Zheng, L., 2012. A comparative study of continuous-time models for scheduling of crude oil operations in inland refineries. Comput Chem Eng 44, 141–167.

Christofides, N., Alvarez-Valdés, R., Tamarit, J. M., 1987. Project scheduling with resource constraints: a branch and bound approach. Eur J Oper Res 29 (3), 262–273.

Collins, J. M., Schmidt, F. L., Sanchez-Ku, M., Thomas, L., McDaniel, M., Le, H., 2003. Can basic individual differences shed light on the construct meaning of assessment center evaluations? Int J Select Assess 11 (1), 17–29.

Grüter, J., Trautmann, N., Zimmermann, A., 2014. An MBLP model for scheduling assessment centers. In: Huisman, D., Louwerse, I., Wagelmans, A. (Eds.), Operations Research Proceedings 2013. Springer, Berlin, pp. 161–167.

Kaplan, L., 1988. Resource-constrained project scheduling with preemption of jobs. Ph.D. thesis, University of Michigan.

Klein, R., 2000. Scheduling of resource-constrained projects. Kluwer, Amsterdam.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., et al., 2011. MIPLIB 2010. Math Prog Comp 3 (2), 103–163.

Kolisch, R., Sprecher, A., 1997. PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. Eur J Oper Res 96 (1), 205–216.

Koné, O., Artigues, C., Lopez, P., Mongeau, M., 2011. Event-based MILP models for resource-constrained project scheduling problems. Comput Oper Res 38 (1), 3–13.

Koné, O., Artigues, C., Lopez, P., Mongeau, M., 2013. Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. Flex Serv Manuf J 25 (1–2), 25–47.

Kopanos, G. M., Kyriakidis, T. S., Georgiadis, M. C., 2014. New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. Comput Chem Eng 68, 96–106.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1998. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Manage Sci 44 (5), 714–729.

Naber, A., Kolisch, R., 2014. MIP models for resource-constrained project scheduling with flexible resource profiles. Eur J Oper Res 239 (2), 335–348.

Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: a zero-one programming approach. Manage Sci 16 (1), 93–108.

Rihm, T., Trautmann, N., 2016. A decomposition approach for an assessment center planning problem. In: Ruiz, R., Alvarez-Valdes, R. (Eds.), Proceedings of the 15th International Conference on Project Management and Scheduling. Valencia, pp. 206–209.

Stefansson, H., Sigmarsdottir, S., Jensson, P., Shah, N., 2011. Discrete and continuous time representations and mathematical models for large production scheduling problems: a case study from the pharmaceutical industry. Eur J Oper Res 215 (2), 383–392.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L. V., 2008. An evaluation of the adequacy of project network generators with systematically sampled networks. Eur J Oper Res 187 (2), 511–524.

Vielma, J. P., 2015. Mixed integer linear programming formulation techniques. SIAM Rev 57 (1), 3–57.

Zapata, J. C., Hodge, B. M., Reklaitis, G. V., 2008. The multimode resource constrained multiproject scheduling problem: alternative formulations. AIChE J 54 (8), 2101–2119.

Zimmermann, A., Trautmann, N., 2014. Scheduling of assessment centers: an application of resource-constrained project scheduling. In: Fliedner, T., Kolisch, R., Naber, A. (Eds.), Proceedings of the 14th International Conference on Project Management and Scheduling. Munich, pp. 263–266.

Zimmermann, A., Trautmann, N., 2015. A list-scheduling approach for the planning of assessment centers. In: Hanzálek, Z., Kendall, G., McCollum, B., Šůcha, P. (Eds.), Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Application. Prague, pp. 541–554.

Paper III

Minimizing operational costs of assessment centers[3]

Tom Rihm

Department of Business Administration
University of Bern

Contents

[3]Rihm, T. (2017). Minimizing operational costs of assessment centers. *In: Kaihara, T., Nonobe, K. (eds.). Proceedings of the International Symposium on Scheduling 2017. Nagoya, 45–50*

Abstract

In an assessment center, candidates for job vacancies perform a set of tasks during which so-called assessors evaluate them. We discuss a novel planning problem that consists of scheduling the tasks and assigning a prescribed number of assessors to the tasks subject to a given deadline and specific assessor-assignment constraints. The objective is to minimize the total operational costs. We develop a mixed-integer linear programming formulation of this problem. To improve the performance, we introduce preprocessing techniques and a row generation scheme. We demonstrate the effectiveness of the proposed approach for a set of test instances derived from real-life data.

3.1 Introduction

Assessment centers (ACs) are used by human resource managers to evaluate candidates' skills and abilities relevant for a job vacancy (cf. Collins et al., 2003). Typically, each candidate performs the same set of predefined tasks, such as presentations, in-basket exercises, structured interviews, and role-play exercises (cf. Spector et al., 2000). Generally, the execution of a task requires a separate room and some assessors (i.e., psychologists or managers) to observe and to evaluate the candidates. The role-play exercises additionally require some actors to simulate a situation that frequently occurs in the vacant position. The operational costs of an AC increase with each individual assessor, actor, and room.

We investigate the assessment center resource investment problem (ACRIP). In the ACRIP, a set of candidates and a set of tasks is given. Each candidate must perform each task exactly once and must have some free time for a lunch break within a given time window. During each task, a prescribed number of assessors and actors must be present. The tasks may include some preparation time for the candidate at the beginning and some evaluation time for the assessors and actors at the end. Typically, the candidates prepare the tasks in a common room. After the preparation time has expired, the assessors, the actors, and the candidate join in a separate room. At the end, the assessors and the actors stay in the room to discuss their observations and to evaluate the candidate. This evaluation time can differ between assessors and actors. The total number of assessors, actors, and rooms that are available is subject to decision. However, the operational costs increase with each additional required assessor, actor, and room. Two assessor-assignment rules constrain the assignment of assessors to the tasks. First, each candidate must be observed by at least half of the total number of assessors rounded down and by at most half

of the total number of assessors rounded up plus one; this supports an objective and fair evaluation of each candidate. Second, no assignment is allowed between candidates and assessors who know each other (no-go relationship). The number of times that an assessor can observe the same candidate is unlimited. The ACRIP consists of a) scheduling all tasks and a lunch break for each candidate and b) assigning a room and the prescribed number of assessors and actors to the tasks such that the above-described constraints and a given deadline (i.e., length of a day) are met. The objective is to minimize the total costs that arise with the required number of assessors, actors, and rooms.

To the best of our knowledge, the assessment center planning problem (ACP) introduced by Grüter et al. (2014) is the only scheduling problem discussed in the context of assessment centers so far. In the ACP, the total duration of the AC is to be minimized while the available number of assessors and actors are given. In contrast to the ACRIP, the rooms and the deadline are not considered. All the other constraints are identical. However, the costs of the AC are affected to a very limited extend only by the duration, because the assessors, actors, and rooms are paid on a daily basis in practice.

In this paper, we present a mixed-integer linear programming (MIP) formulation for the ACRIP. We tighten the formulation by introducing some preprocessing techniques. Furthermore, we propose a novel row generation scheme that exploits the structural properties of the ACRIP to speed up the search process. Row generation has been known for a long time and has recently shown promising results for various planning problems (cf., e.g., Della Croce et al., 2017; Pferschy and Staněk, 2017). In our row generation scheme, some constraints that mainly drive the computation time of a general-purpose solver are dropped and only provided to the solver at runtime if necessary. Each time the solver finds a feasible integer solution that violates one of the dropped constraints, a violated constraint is added to the formulation and a MIP-based heuristic attempts to transform the solution into a feasible solution for the ACRIP. In a computational study, we test the MIP formulation with and without the row generation scheme on a set of instances derived from real-life data. The results show that the use of the row generation scheme increases the number of optimal solutions which are found within a prescribed time limit considerably.

The remainder of this paper is organized as follows. In Section 3.2, we describe the ACRIP in more detail. In Section 3.3, we discuss the related literature. In Section 3.4, we describe the solution approach. In Section 3.5, we report the computational results. In Section 3.6, we conclude the paper with a summary and some directions for future research.

Table 3.1: Data of the illustrative example

	E_1	E_2	E_3	Lunch
Activity related to candidate C_1	1	4	7	10
Activity related to candidate C_2	2	5	8	11
Activity related to candidate C_3	3	6	9	12
Required number of assessors	2	1	2	0
Required number of actors	1	1	0	0
Total duration	14	11	9	6
Duration of preparation time (candidates)	4	3	0	0
Duration of evaluation time (assessors)	4	3	2	0
Duration of evaluation time (actors)	4	2	0	0

3.2 Illustration of the ACRIP

In this section, we illustrate the ACRIP with an example that comprises three candidates (C_1, C_2, C_3) and three tasks (E_1, E_2, E_3). The deadline to complete the AC is 40. Table 3.1 lists the data. Each candidate's tasks and lunch break represent individual activities, e.g., activity 1 corresponds to task E_1 of candidate C_1. Hence, 12 activities are considered in total. Activities 1–3 require each 2 assessors, 1 actor, and they have a total duration of 14 time units including 4 time units for the preparation and the evaluation, respectively. The number of assessors, actors, and rooms that are available is not limited. The example does not include candidates and assessor with a no-go relationship. The costs for one assessor, actor, and room are 1,000, 300, and 500, respectively. Finally, the earliest and latest start times for the lunch breaks are 15 and 29, respectively.

Figure 3.1 presents an optimal solution for the illustrative example. This solution includes 4 assessors (A_1–A_4), 2 actors (P_1, P_2), and 3 rooms (R_1–R_3). Hence, the total costs are $4{\cdot}1{,}000+2{\cdot}300+3{\cdot}500=6{,}100$. The assessor-assignment rules are met, because each candidate is observed by 2 or 3 different assessors.

3.3 Related literature

In this section, we review planning problems discussed in the literature that are similar to the ACRIP. In Subsection 3.3.1, we review the existing solution methods for the ACP. In Subsection 3.3.2, we review two related planning problems that arise in the context of project scheduling. The ACRIP is also related to planning problems such as assigning

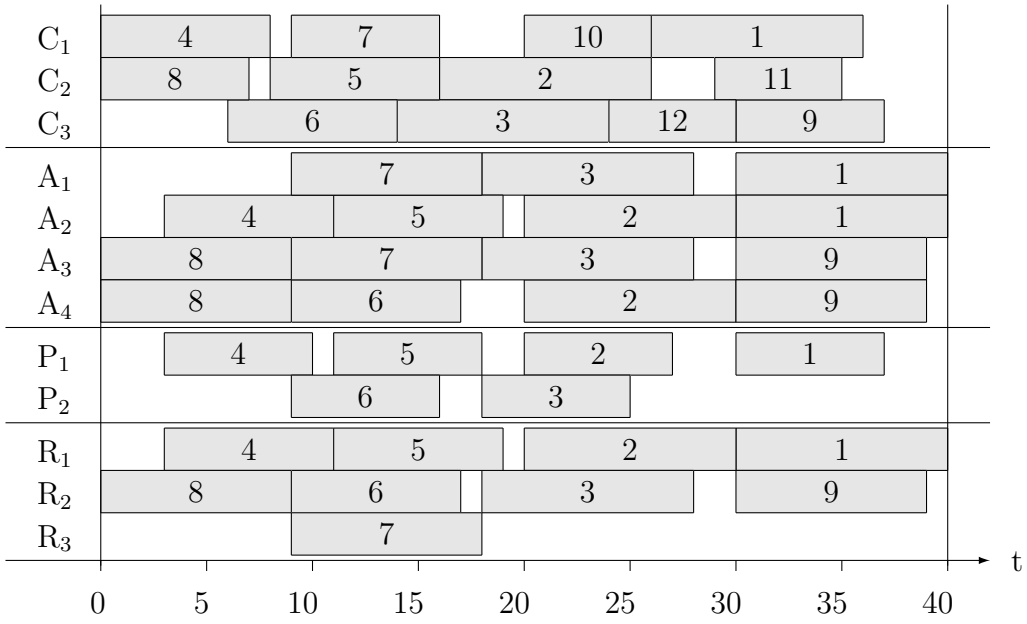


Figure 3.1: Optimal schedule of the illustrative example

shifts to a multi-skilled workforce (cf., e.g., Krishnamoorthy et al., 2012) and scheduling operating rooms (cf., e.g., Cardoen et al., 2010).

3.3.1 Assessment center scheduling

The ACP originates from a human resource management service provider that organizes ACs regularly for other companies, and is first described by Grüter et al. (2014). Rihm et al. (2016) compare the performance of two discrete-time (DT) and three continuous-time (CT) MIP formulations to solve the ACP. In DT formulations, the activities can only start at predefined time points. The duration between the time points reflects the planning accuracy, which is typically around 5 minutes in the ACP. Conversely, in CT formulations, the activities can start at any time point. The results show that the CT formulations provide better solutions within limited computation time; however, the best MIP-based lower bounds are obtained with the DT formulations.

Heuristic procedures are presented by Zimmermann and Trautmann (2015) and Rihm and Trautmann (2016). Zimmermann and Trautmann (2015) present a multi-pass list scheduling heuristic, which is based on a serial schedule generation scheme. Rihm and Trautmann (2016) develop a heuristic that decomposes the ACP into a scheduling and an assignment subproblem. The scheduling subproblem determines the start times of all tasks and the lunch break for each candidate, whereas the assignment subproblem defines which assessors are assigned to which candidate during each task. The results show that

this decomposition heuristic outperforms the above-mentioned approaches. However, this method does not guarantee to find an optimal solution.

In this paper, we present a DT formulation for the ACRIP because DT formulations yield the best lower bounds. To speed up the search process of a general-purpose solver, we present a row generation scheme that also separates the scheduling and the assignment decisions.

3.3.2 Project scheduling

The ACRIP is similar to various project scheduling problems discussed in the literature. A project is characterised by a set of activities, which require certain amounts of scarce renewable resources during their execution. The planning problem consists in scheduling the activities subject to prescribed completion-start precedence and renewable-resource constraints. In the ACRIP, each candidate's tasks and lunch break can be considered as a project activity, and the candidates, assessors, actors, and rooms represent renewable resources. However, owing to the specific assessor-assignment rules, solution approaches for project scheduling problems are not directly applicable to the ACRIP.

In the context of the project scheduling literature, a similar distinction between the objective functions is made. On the one hand, the minimization of the total duration for given resource capacities is referred to as the *resource-constrained project scheduling problem* (cf., e.g., Artigues, 2010), and on the other hand, the minimization of the total resource costs subject to a prescribed deadline for the project completion is referred to as the *resource investment problem* (cf., e.g., Möhring, 1984). For a comprehensive overview of the different project scheduling problems, we refer to Hartmann and Briskorn (2010).

3.4 Exact approach

In this section, we discuss our solution methods in detail. We introduce the notation in Subsection 3.4.1, present preprocessing techniques in Subsection 3.4.2, establish the MIP formulation in Subsection 3.4.3, and describe the row generation scheme in Subsection 3.4.4.

3.4.1 Notation

Tables 3.2 and 3.3 provide the notation used in this paper.

Table 3.2: Sets and parameters of the MIP formulations

A	Set of available assessors
C	Set of candidates
I	Set of activities (including lunch breaks)
I^A	Set of activities that require at least one assessor
I^P	Set of activities that require at least one actor
I_c	Set of activities that require candidate $c \in C$
I^L	Set of lunch breaks
N	Set of candidate-assessor pairs (c, a) with a no-go relationship
k^A	Cost for an assessor
k^P	Cost for an actor
k^R	Cost for a room
e_i	Earliest start time for activity i
l_i	Latest start time for activity i
$\overline{n}^A$	Upper bound on the number of assessors required
$\underline{n}^A$	Lower bound on the number of assessors required
$\underline{n}^P$	Lower bound on the number of actors required
$\underline{n}^R$	Lower bound on the number of rooms required
p_i	Total duration of activity i (including preparation and execution times)
p_i^C	Preparation time of activity i for candidates
p_i^A	Evaluation time of activity i for assessors
p_i^P	Evaluation time of activity i for actors
r_i^A	Number of assessors required by activity i
r_i^P	Number of actors required by activity i
T	Deadline for the duration of the assessment center

3.4.2 Preprocessing

In this subsection, we devise upper and lower bounds on the required number of assessors, actors, and rooms, respectively. An upper bound on the number of assessors is necessary to define the set of available assessors. An upper bound on the number of assessors follows from the assessor-assignment rules. Because each candidate must be observed by at least half of the total number of assessors rounded down, it follows that the maximum number

Table 3.3: Variables of the MIP formulation

n^A	Total number of assessors required
n^P	Total number of actors required
n^R	Total number of rooms required
V_{ca}	$\begin{cases} = 1, \text{ if assessor } a \text{ is assigned to candidate } c \text{ at least once} \\ = 0, \text{ otherwise} \end{cases}$
W_a	$\begin{cases} = 1, \text{ if assessor } a \text{ is assigned to at least one candidate} \\ = 0, \text{ otherwise} \end{cases}$
X_{it}	$\begin{cases} = 1, \text{ if activity } i \text{ starts at time point } t \\ = 0, \text{ otherwise} \end{cases}$
Y_{ij}	$\begin{cases} = 1, \text{ if activity } i \text{ is performed before } j \neq i \text{ by the assessors} \\ = 0, \text{ otherwise} \end{cases}$
Z_{ia}	$\begin{cases} = 1, \text{ if assessor } a \text{ is assigned to activity } i \\ = 0, \text{ otherwise} \end{cases}$

of assessors cannot exceed $\overline{n}^A$.

$$\overline{n}^A = \min_{c \in C} \left(2 \sum_{i \in I_c} r_i^A \right) + 1 \tag{3.1}$$

A lower bound for the number of assessors ($\underline{n}^A$) that are required to run the AC corresponds to the maximum number of assessors required by an activity. Another lower bound is obtained by dividing the total workload of the assessors by their total time available. A similar lower bound is used by Drexl and Kimms (2001) for the resource investment problem. The total time that is available for the assessors corresponds to the deadline minus the shortest preparation time of an activity, because the assessors cannot start before that time.

$$\underline{n}^A = \max \left(\max_{i \in I^A} r_i^A, \left\lceil \frac{\sum_{i \in I^A} r_i^A (p_i - p_i^C)}{T - \min_{i \in I^A} p_i^C} \right\rceil \right) \tag{3.2}$$

Analogously, we define a lower bound on the number of actors ($\overline{n}^P$) and rooms ($\overline{n}^R$),

respectively.

$$\underline{n}^P = \max\left(\max_{i\in I^P} r_i^P, \left\lceil \frac{\sum_{i\in I^P} r_i^P (p_i - p_i^C - p_i^A + p_i^P)}{T - \min_{i\in I^P} p_i^C} \right\rceil\right) \tag{3.3}$$

$$\underline{n}^R = \left\lceil \frac{\sum_{i\in I^A} (p_i - p_i^C)}{T - \min_{i\in I^A} p_i^C} \right\rceil \tag{3.4}$$

3.4.3 MIP formulation

The MIP formulation is based on the discrete-time formulation that performs the best in the comparative analysis of Rihm et al. (2016) for the ACP. We use the binary variables X_{it}, which are equal to 1 if activity $i \in I$ starts at time $t = e_i, \ldots, l_i$. To model the assessor-assignment rules, we additionally use three types of binary variables: the sequencing variables Y_{ij}, the activity-assignment variables Z_{ia}, and the candidate-assignment variables V_{ca}.

The objective is to minimize the total costs of the AC

$$k^A n^A + k^P n^P + k^R n^R. \tag{3.5}$$

Constraints (3.6) state that each activity starts once. The earliest and latest start times (e_i and l_i) are required to model the time windows for the lunch break activities ($i \in I^L$). For the other activities ($i \in I \backslash I^L$), we set $e_i = 0$ and $l_i = T - p_i$. This ensures that each activity is completed within the deadline T.

$$\sum_{t=e_i}^{l_i} X_{it} = 1 \qquad (i \in I) \tag{3.6}$$

Constraints (3.7) prevent that any candidate c performs more than one activity at the same time t. Because the evaluation time for the assessors and actors is included in the total duration, activity i is performed at time t by a candidate if activity i starts between time $t + 1 - p_i + p_i^A$ and t.

$$\sum_{i\in I_c} \sum_{s=\max(e_i, t+1-p_i+p_i^A)}^{\min(l_i,t)} X_{is} \leq 1 \qquad (c \in C;\ t = 0, \ldots, T) \tag{3.7}$$

Analogously, constraints (3.8)–(3.10) ensure that only activities that do not require more than the determined number of assessors, actors, and rooms are executed at the same

time.

$$\sum_{i \in I^A} \sum_{s=\max(e_i, t+1-p_i)}^{\min(l_i, t-p_i^C)} r_i^A X_{is} \leq n^A \qquad (t = 0, \ldots, T) \tag{3.8}$$

$$\sum_{i \in I^P} \sum_{s=\max(e_i, t+1-p_i+p_i^A-p_i^P)}^{\min(l_i, t-p_i^C)} r_i^P X_{is} \leq n^P \qquad (t = 0, \ldots, T) \tag{3.9}$$

$$\sum_{i \in I^A} \sum_{s=\max(e_i, t+1-p_i)}^{\min(l_i, t-p_i^C)} r_i^A X_{is} \leq n^R \qquad (t = 0, \ldots, T) \tag{3.10}$$

Without the assessor-assignment rules, constraints (3.6)–(3.10) would be sufficient to model the ACRIP. The following constraints only address the assessor-assignment rules. Constraints (3.11) link the start time variables to the sequencing variables.

$$\sum_{t=e_i}^{l_i} (t + p_i) X_{it} \leq \sum_{t=e_j}^{l_j} (t + p_j^C) X_{jt} + T(1 - Y_{ij}^A) \qquad (i, j \in I^A : \ i \neq j) \tag{3.11}$$

Constraints (3.12) ensure that the required number of assessors are assigned to each activity.

$$\sum_{a \in A} Z_{ia} = r_i^A \qquad (i \in I^A) \tag{3.12}$$

Constraints (3.13) require that $W_a = 1$ if assessor a is assigned to any activity.

$$Z_{ia} \leq W_a \qquad (i \in I^A, \ a \in A) \tag{3.13}$$

Constraints (3.14) state that the total number of assessors required is equal to the sum of all assessors that are assigned to at least one activity.

$$\sum_{a \in A} W_a = n^A \qquad (a \in A) \tag{3.14}$$

Constraints (3.15) link the activity-assignment variables to the sequencing variables. If assessor a is assigned to activities i and j, then activities i and j must be performed in

sequence.

$$Y_{ij} + Y_{ji} \geq Z_{ia} + Z_{ja} - 1 \qquad (i, j \in I^A,\ a \in A:\ i < j) \tag{3.15}$$

Constraints (3.16) forbid cycles in the sequencing decisions.

$$Y^A_{ij} + Y^A_{ji} \leq 1 \qquad (i, j \in I^A:\ i < j) \tag{3.16}$$

Constraints (3.17) address the assessor-assignment rules, i.e., each candidate must be observed by at least half of the total number of assessors rounded down and by at most half of the total number of assessors rounded up plus one.

$$0.5n^A - 0.5 \leq \sum_{a \in A} V_{ca} \leq 0.5n^A + 1.5 \qquad (c \in C) \tag{3.17}$$

Constraints (3.18) link the candidate-assignment variables V_{ca} to the activity-assignment variables Z^A_{ia}, i.e., $V_{ca} = 1$ if and only if assessor a is assigned to at least one activity that requires candidate c.

$$\sum_{i \in I_c \setminus I^L} \frac{Z^A_{ia}}{|I_c \setminus I^L|} \leq V_{ca} \leq \sum_{i \in I_c \setminus I^L} Z^A_{ia} \qquad (c \in C,\ a \in A) \tag{3.18}$$

Constraints (3.19) model the no-go relationships.

$$V_{ca} = 0 \qquad ((c, a) \in N) \tag{3.19}$$

In sum, formulation (F) reads as follows.

$$
(\mathrm{F}) \quad \begin{cases}
\text{Min} & k^A n^A + k^P n^P + k^R n^R & \\
\text{s.t.} & (3.6)\text{–}(3.19) & \\
& \underline{n}^A \leq n^A \in \mathbb{N} & \\
& \underline{n}^P \leq n^P \in \mathbb{N} & \\
& \underline{n}^R \leq n^R \in \mathbb{N} & \\
& V_{ca} \in \{0,1\} & (c \in C,\ a \in A) \\
& W_a \in \{0,1\} & (a \in A) \\
& X_{it} \in \{0,1\} & (i \in I;\ t = e_i, \ldots, l_i) \\
& Y_{ij} \in \{0,1\} & (i,j \in I^A :\ i \neq j) \\
& Z_{ia} \in \{0,1\} & (i \in I^A,\ a \in A)
\end{cases}
$$

3.4.4 Row generation scheme

In this subsection, we present a row generation scheme for formulation (F) to improve the performance of a general-purpose MIP solver. The scheme exploits the fact that the scheduling and the assignment subproblem of the ACRIP are solved much faster if they are solved independently. For this reason, the linking constraints (3.11) are dropped and only provided to the solver at runtime if necessary.

For overview purposes, the row generation scheme is summarized as a flowchart in Figure 3.2. The solver starts the branch-and-bound process without considering the linking constraints. Whenever an integer solution (S^-) is found, we check if solution S^- is feasible for the ACRIP, i.e., if all linking constraints are met. If so, we accept S^- as a new incumbent solution and the solver continues with the branch-and-bound process. If not, we return one violated linking constraint to the solver, which will thereupon discard solution S^-. We select that violated linking constraint for which the corresponding pair of activities overlap for assessors the most. Notably, we only have to check the linking constraints for all pairs of activities that overlap for assessors in solution S^-.

If solution S^- is not feasible for the ACRIP, we attempt to build a feasible solution with the following MIP-based heuristic. We fix the start time variables X_{it} and solve a reduced model that contains the objective function (3.5) and the constraints (3.11)–(3.19). In general, this reduced problem is solved to optimality or is shown to be infeasible within very short computation time. If the heuristic devises a solution that is feasible for the ACRIP, we return this solution to the solver and the solver continues with the branch-and-bound process.

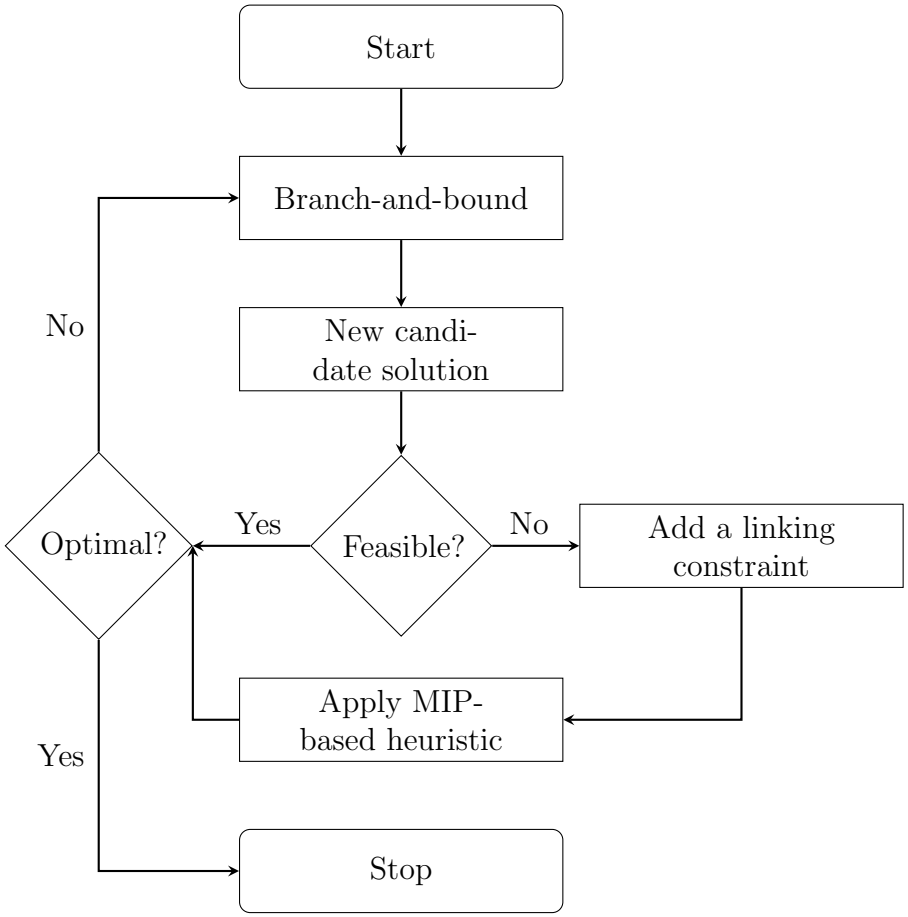


Figure 3.2: Overview of the row generation scheme

3.5 Computational analysis

In this section, we evaluate the performance of the proposed approaches. We implemented the MIP formulation (F) and the row generation scheme (R) in Python 3.5, and we used the Gurobi Optimizer 7.0 with the default solver settings for the optimization. We limited the computation time of the solver to 600 seconds. All computations were performed on a workstation equipped with two 6-core Intel(R) Xeon(R) X5650 CPUs running at 2.66 GHz, and with 24 GB RAM.

In Subsection 3.5.1, we present the test instances used. In Subsection 3.5.2, we provide the numerical results.

3.5.1 Test instances

For the computational analysis, we adapt two test sets from the ACP. The first set contains the four real-life instances presented by Grüter et al. (2014), and the second set contains

the 240 systematically generated instances presented by Rihm et al. (2016). In the first set, all instances contain the same five tasks and only the number of candidates ($= 6, 7, 9, 11$) varies. In the second set, each instance contains four or five tasks and between four and eleven candidates. We drop the number of assessors and actors, which are included in both test sets, because these numbers are to be determined in the ACRIP. The set of available assessors A contains all assessors of the corresponding ACP-instance (including their no-go relationships) and some additional assessors (without no-go relationships) such that $|A| = \overline{n}^A$. $\overline{n}^A$ corresponds to the upper bound on the number of assessors presented in Section 3.4.2. Without the additional assessors, a feasible solution may not exist if the deadline is short.

For all the instances, we set the costs to $k^A = 1,000$, $k^P = 300$, and $k^R = 500$. To compute the deadline T, we proceed as described by Drexl and Kimms (2001) for the resource investment problem. We compute a lower bound for the duration of the AC and multiply it by a parameter (Θ). Because all activities that require the same candidate are performed sequentially, the sum of the durations of these activities corresponds to a lower bound for the total duration. Hence, deadline T is computed as follows.

$$T = \Theta \max_{c \in C} \left(\sum_{i \in I_c} (p_i - p_i^A) \right) \tag{3.20}$$

For the real-life instances, we use $\Theta = 1.2, 1.3, \ldots, 1.8$. For the test instances, we use $\Theta = 1.2, 1.4, 1.6, 1.8$.

3.5.2 Numerical results

Table 3.4 lists the results for the four real-life instances (RL1,...,RL4) and $\Theta = 1.4$. We compare the required computation times in seconds (CPU), the objective function values (OFV) and the lower bounds (LB) obtained by the solver for formulation (F) and the row generation scheme (R). For each instance, the best objective function values obtained are highlighted in boldface. In contrast to formulation (F), the row generation scheme solves all four instances to optimality within the time limit.

Solving the ACRIP with several deadlines gives an insight into the tradeoff between the duration and the total costs of the AC. Figure 3.3 shows the relative cost savings if the deadline is varied. As benchmark, we use $\Theta = 1.2$. The solutions are obtained with the row generation scheme. Increasing the deadline enables cost savings of up to 40%.

Table 3.5 summarizes the results for the test set. The row generation scheme (R) provides feasible solutions to all instances, whereas formulation (F) fails to find a feasible

Table 3.4: Numerical results for real-life instances and $\Theta = 1.4$

Instance	F			R		
	OFV	LB	CPU	OFV	LB	CPU
RL1	11,600	10,600	600	**10,600**	10,600	17
RL2	25,700	13,100	600	**13,100**	13,100	395
RL3	13,400	13,100	600	**13,100**	13,100	116
RL4	8,800	7,800	600	**7,800**	7,800	38

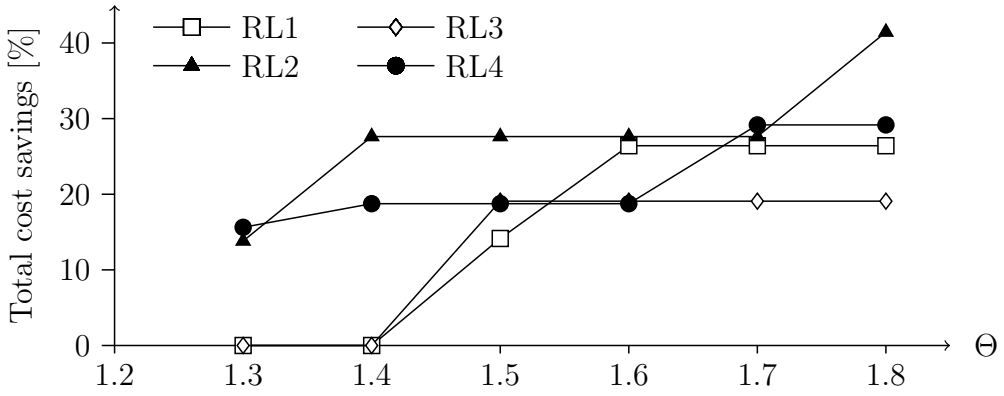


Figure 3.3: Total cost savings for the real-life instances

solution for 31 instances. In total, the row generation scheme is able to devise much more optimal solutions than formulation (F). For this reason, formulation (F) takes on average more than twice the time of the row generation scheme.

To calculate the relative gaps between the solutions and lower bounds obtained by the solver, we use the formula $((OFV - LB)/LB)$. The average gap obtained with the row generation scheme is 1.4% for $\Theta = 1.2$ and decreases as the value of Θ increases. Hence, the level of factor Θ has an impact on the performance of formulation (F). For formulation (F), such an effect is not observed; the average gap is 13.3% for $\Theta = 1.2$ and 15.2% for $\Theta = 1.8$.

3.6 Conclusions and outlook

In this study, we considered scheduling assessment centers with minimal operational costs. Up to now, only solution approaches had been proposed that target a minimal duration. In practice, however, the personnel and the rooms required for the assessment process are often paid on a daily basis. Hence, reducing the duration of the assessment center does

Table 3.5: Numerical results for test instances

		Θ				All
		1.2	1.4	1.6	1.8	
Number of feasible solutions	F	218	231	240	240	929
	R	240	240	240	240	960
Number of optimal solutions	F	65	83	89	108	345
	R	182	206	225	226	839
Average computation time [s]	F	481	436	411	384	428
	R	219	141	95	80	134
Average gap [%]	F	13.3	13.6	13.4	15.2	13.9
	R	1.4	1.0	0.8	0.7	1.0

not affect the costs. For this reason, we proposed to minimize the costs for the personnel and rooms directly subject to a given deadline for the total duration. We developed a mixed-integer programming formulation and proposed a row generation scheme to speed up the search process. Our computational results validate the effectiveness of the row generation scheme on a set of instances from the literature.

A promising idea for future research is to eliminate some symmetric solutions from the search space, because the number of symmetric solutions of this problem may slow down a general-purpose solver. Furthermore, it will be interesting to apply the row generation scheme to related planning problems from the literature such as the multi-skill project scheduling problem (cf., e.g., Bellenguez-Morineau and Néron, 2007).

Bibliography

Artigues, C., 2010. The resource-constrained project scheduling problem. In: Artigues, C., Demassey, S., Néron, E. (Eds.), Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE Ltd and John Wiley & Sons, Inc., London, pp. 19–35.

Bellenguez-Morineau, O., Néron, E., 2007. A branch-and-bound method for solving multi-skill project scheduling problem. RAIRO-Operations Research-Recherche Opérationnelle 41 (2), 155–170.

Cardoen, B., Demeulemeester, E., Beliën, J., 2010. Operating room planning and scheduling: A literature review. European Journal of Operational Research 201 (3), 921–932.

Collins, J. M., Schmidt, F. L., Sanchez-Ku, M., Thomas, L., McDaniel, M., Le, H., 2003. Can basic individual differences shed light on the construct meaning of assessment center evaluations? International Journal of Selection and Assessment 11 (1), 17–29.

Della Croce, F., Koulamas, C., T'kindt, V., 2017. A constraint generation approach for two-machine shop problems with jobs selection. European Journal of Operational Research 259 (3), 898–905.

Drexl, A., Kimms, A., 2001. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society 52 (3), 340–351.

Grüter, J., Trautmann, N., Zimmermann, A., 2014. An MBLP model for scheduling assessment centers. In: Huisman, D., Louwerse, I., Wagelmans, A. (Eds.), Operations Research Proceedings 2013. Springer, Berlin, pp. 161–167.

Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207 (1), 1–14.

Krishnamoorthy, M., Ernst, A. T., Baatar, D., 2012. Algorithms for large scale shift minimisation personnel task scheduling problems. European Journal of Operational Research 219 (1), 34–48.

Möhring, R. H., 1984. Minimizing costs of resource requirements in project networks subject to a fixed completion time. Operations Research 32 (1), 89–120.

Pferschy, U., Staněk, R., 2017. Generating subtour elimination constraints for the TSP from pure integer solutions. Central European Journal of Operations Research 25 (1), 231–260.

Rihm, T., Trautmann, N., 2016. A decomposition approach for an assessment center planning problem. In: Ruiz, R., Alvarez-Valdes, R. (Eds.), Proceedings of the 15th International Conference on Project Management and Scheduling. Valencia, pp. 206–209.

Rihm, T., Trautmann, N., Zimmermann, A., 2016. MIP formulations for an application of project scheduling in human resource management. Flexible Services and Manufacturing Journal, in press.

Spector, P. E., Schneider, J. R., Vance, C. A., Hezlett, S. A., 2000. The relation of cognitive ability and personality traits to assessment center performance. Journal of Applied Social Psychology 30 (7), 1474–1491.

Zimmermann, A., Trautmann, N., 2015. A list-scheduling approach for the planning of assessment centers. In: Hanzálek, Z., Kendall, G., McCollum, B., Šůcha, P. (Eds.), Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Application. Prague, pp. 541–554.

Paper IV

Staff assignment with lexicographically ordered acceptance levels[4]

Tom Rihm Philipp Baumann

Department of Business Administration
University of Bern

Contents

[4]Rihm, T., Baumann, P. (2018). Staff assignment with lexicographically ordered acceptance levels. *Journal of Scheduling 21(2), 167-189.* DOI: 10.1007/s10951-017-0525-1

Abstract

Staff assignment is a compelling exercise that affects most companies and organizations in the service industries. Here, we introduce a new real-world staff assignment problem that was reported to us by a Swiss provider of commercial employee scheduling software. The problem consists of assigning employees to work shifts subject to a large variety of critical and noncritical requests, including employees' personal preferences. Each request has a target value, and deviations from the target value are associated with integer acceptance levels. These acceptance levels reflect the relative severity of possible deviations, e.g., for the request of an employee to have at least two weekends off, having one weekend off is preferable to having no weekend off and thus receives a higher acceptance level. The objective is to minimize the total number of deviations in lexicographical order of the acceptance levels. Staff assignment approaches from the literature are not applicable to this problem. We provide a binary linear programming formulation and propose a matheuristic for large-scale instances. The matheuristic employs effective strategies to determine the subproblems and focuses on finding good feasible solutions to the subproblems rather than proving their optimality. Our computational analysis based on real-world data shows that the matheuristic scales well and outperforms commercial employee scheduling software.

4.1 Introduction

Employee scheduling problems arise in hospitals, banks, hotels, police stations, companies in the service industry, and other organizations. In their general form, employee scheduling problems involve a) the determination of shift types, b) the temporal scheduling of shifts, and c) the assignment of employees to shifts. For a comprehensive overview of employee scheduling problems, we refer to Ernst et al. (2004), Van den Bergh et al. (2013), and De Bruecker et al. (2015). We focus here on the assignment of employees to shifts after the shift types and their start times have been determined. In most real-world applications, the assignment of employees to shifts is a challenging task because a large variety of critical and noncritical requests must be considered. Critical requests pertain to work laws and policies imposed by the management and must be accepted to obtain a feasible assignment. Noncritical requests are usually related to employee preferences and

can be refused in feasible assignments. However, accepting noncritical requests increases employee satisfaction, which in turn positively affects productivity and eventually results in high customer satisfaction. In practice, most employee scheduling software packages model the trade-offs between noncritical requests based on user-defined weights. This places a heavy burden on the user because she or he is required to repeatedly adjust the weights of the noncritical requests until a satisfactory solution is obtained (cf., e.g., Parr and Thompson, 2007). Only if relevant historical data is available in the form of past schedules, the configuration of these weights can be partially automated (cf., e.g., Mihaylov et al., 2016).

The planning problem considered in this paper stems from a Swiss provider of employee scheduling software who has developed a new user interface to specify trade-offs among noncritical requests that is not based on user-defined weights. The user defines a target value for each request and assigns integer acceptance levels (AL) from the set $\{0,1,\ldots,100\}$ to deviations from this target value. The acceptance levels are ordered lexicographically, i.e., a deviation associated with a lower acceptance level is considered more important than any number of deviations associated with higher acceptance levels. This framework has received positive customer feedback because the user has an intuitive understanding of how the specified inputs affect the final solution. The framework gives rise to a new type of staff assignment problem that we refer to as the staff assignment problem with lexicographically ordered acceptance levels (SAP-LAL). The objective in the SAP-LAL is to minimize the total number of deviations in lexicographical order of the acceptance levels.

The literature on exact approaches for employee scheduling problems with multiple and conflicting requests concentrates on goal programming approaches (e.g., Beaulieu et al., 2000; Azaiez and Al Sharif, 2005; Topaloglu, 2006; Al-Yakoob and Sherali, 2007; Eiselt and Marianov, 2008; Falasca et al., 2011; Louly, 2013). In goal programming, which was introduced by Charnes et al. (1955), each request is assigned a target value, and deviations from the target values are minimized. Goal programming approaches are based on mathematical programs and thus provide great flexibility to accommodate a large variety of requests. The most widely used variants of goal programming are weighted and lexicographic goal programming (cf., e.g., Tamiz et al., 1995). Weighted goal programming approaches are not applicable to the SAP-LAL because the range of weights required to ensure that less-accepted deviations are always minimized before more-accepted deviations grows rapidly with the number of different acceptance levels and may become large enough to cause numerical problems for solvers. Existing lexicographic goal programming approaches minimize deviations sequentially and have therefore only

been designed for applications with a predefined ranking of requests. Such a predefined ranking is not given in the SAP-LAL. Furthermore, despite improvements in optimization software and computer hardware, the performance of exact goal programming approaches is still insufficient for large-scale problem instances.

For large-scale instances, various heuristics have been proposed. Among those methods, matheuristics have recently shown promising results (cf. Smet and Vanden Berghe, 2012; Della Croce and Salassa, 2014; Smet et al., 2014b). Matheuristics decompose the original problem into subproblems which are then solved using a mathematical program (cf., e.g. Raidl and Puchinger, 2008; Boschetti et al., 2009; Maniezzo et al., 2009; Ball, 2011). Hence, they combine the flexibility of mathematical programs to easily accommodate complex constraints with the ability of heuristics to find good solutions quickly. The performance of matheuristics strongly depends on the construction of the subproblems. Existing matheuristics for employee scheduling problems either use purely random strategies for constructing the subproblems (cf. Smet and Vanden Berghe, 2012), or construct the subproblems such that the corresponding mathematical programs are as large as the programs for the original problems in terms of constrains and variables (cf. Della Croce and Salassa, 2014; Smet et al., 2014b). The existing matheuristics are therefore not appropriate for large-scale SAP-LAL instances because for those instances it is crucial that the subproblems focus on the decisions which directly impact the quality of the solution and that the corresponding mathematical programs are small.

In this paper, we propose a new strategy for decomposing the requests into subrequests which allows us to formulate the SAP-LAL as a lexicographic goal program. Despite the decomposition, the resulting lexicographic goal program constitutes an exact solution approach. To reduce the size of the program, we propose novel aggregation techniques. For large-scale instances, we develop, based on the lexicographic goal program, a matheuristic that iteratively improves an initial feasible solution by reassigning specific subsets of employees. The main methodological feature of the matheuristic is an employee selection rule for constructing the subproblems effectively. The rule selects, for each subproblem, a subset of employees such that at least one employee in the subset has a refused request and that this refusal can be eliminated by a swap of shifts with at least one other employee in the subset. This rule differentiates the proposed matheuristic from existing matheuristics as it ensures that the subproblems focus on the decisions which directly impact the quality of the solution. Moreover, since the number of employees is the main driver of the problem size, the selection rule allows to write small and compact mathematical programs for the subproblems that involve only the selected employees. In contrast to existing matheuristics (cf., e.g., Della Croce and Salassa, 2014), which

obtain the subproblems by fixing the values of some of the decision variables of the complete model, our strategy significantly reduces the number of redundant constraints and variables of the respective models and thus improves running times.

In a computational analysis, we apply the exact approach and the matheuristic to a real-world instance and a test set that contains 45 instances derived from real-world data. The exact approach finds optimal solutions for small- and medium-sized instances, and the matheuristic delivers high-quality solutions for large-scale instances with limited computational effort. The matheuristic even outperforms a commercial employee scheduling software that is tailored to the SAP-LAL. It turns out that it is beneficial to run the matheuristic in an eager manner, i.e., to impose a short time limit for the solution of the subproblems. This setup of the matheuristic exploits that optimal solutions of the subproblems are often found within a few seconds, while most of the time is spent on proving the optimality of this solution. This finding is of general interest for the development of matheuristics, independent of the context.

The remainder of the paper is structured as follows. In Section 4.2, we formally introduce the SAP-LAL and provide an illustrative example. In Section 4.3, we review the literature on employee scheduling. In Sections 4.4 and 4.5, we describe the exact solution approach and the matheuristic, respectively. In Section 4.6, we report the design and the results of our computational analysis. Section 4.7 concludes the paper with a summary and directions for future research.

4.2 Staff assignment problem with lexicographically ordered acceptance levels

The staff assignment problem with lexicographically ordered acceptance levels (SAP-LAL) was reported to us by a Swiss provider of employee scheduling software. The problem stems from an online tool that currently supports many companies and organizations in assigning employees to work shifts. We describe the SAP-LAL formally in Section 4.2.1 and provide an illustrative example in Section 4.2.2.

4.2.1 Problem description

Consider a set of employees, a set of work shifts with predefined start times and durations, and a set of critical and noncritical requests. Each employee possesses a specific set of skills. There is no difference in skill level and the skills of an employee determine which shifts he or she can perform. When an employee has a skill set that allows her or him

Table 4.1: Types of requests

Type	Description	Critical?
1.	At most one shift per employee and day	yes
2.	Exactly one employee is assigned to each shift	yes
3.	Only employees with the required skills can be assigned to a shift	yes
4.	Preferences for days off should be considered	no
5.	Avoid more than 5 consecutive work days	no
6.	No isolated days off	no
7.	11 hours rest between consecutive shifts	no
8.	Either zero or two shifts on weekends	no
9.	Lower bound on number of weekends off	no
10.	Workload should not exceed target	no
11.	Workload should not be below target	no
12.	No. of early shifts should not exceed target	no
13.	No. of late shifts should not exceed target	no

to perform more than one shift, she or he actually possesses all separate skills to perform each single shift. Hence, according to the classification of De Bruecker et al. (2015), the skills are of the categorical type.

Table 4.1 provides a list of the 13 types of critical and noncritical requests considered in this paper. According to the software provider, these 13 types of requests are sufficient to cover the modeling needs of most companies. For a comprehensive list of other requests that frequently occur in the literature, we refer to Van den Bergh et al. (2013). Critical requests pertain to work laws, contract specifications, and the availability and skills of employees and must therefore be accepted to obtain a feasible assignment. The critical requests are as follows. First, an employee can be assigned to at most one shift per day. Second, each shift requires exactly one employee. This means that if several employees work at the same time, multiple shifts will run in parallel. Third, each shift requires a specific set of skills, and only employees with these skills can be assigned to the corresponding shift. Noncritical requests concern employees' personal preferences and can be refused in a feasible assignment. Among the noncritical requests presented in Table 4.1, requests of type 6 might be less known. The main purpose of requests of type 6 is to prevent so-called *on-off-on* work patterns. In an *on-off-on* work pattern, the employee works on day d, has day $d+1$ off, and works again on day $d+2$. Employees usually prefer

on-off-off-on work patterns. For example, if an employee works on 5 of 7 days, she or he generally prefers to have two consecutive days off (e.g. Thursday and Friday) rather than two isolated (non-consecutive) days off (e.g. Tuesday and Friday). Requests of type 6 allow to express this preference.

Usually, multiple requests of the same type are specified in an instance of the problem. For example, the lower bound on the number of weekends off can be specified individually for different employees. Hence, each noncritical request has an individual target value. In addition, for each noncritical request, a piecewise-constant function maps deviations from the target value to integer acceptance levels from the set $\{0,1,\ldots,100\}$. These mapping functions are defined by the scheduler in consultation with the employees. An acceptance level of zero indicates that the corresponding deviation is unacceptable and leads to an infeasible assignment. An acceptance level of 100 indicates that the target value or an even better value was achieved. Figure 4.1 shows two possible mapping functions for a request of type 9. Figure 4.1(a) corresponds to a request that an employee has at least two weekends off during the planning horizon. Hence, if the employee has two or more weekends off, an acceptance level of 100 is achieved. Having only one weekend off is associated with an acceptance level of 80, and having no weekend off is associated with an acceptance level of 50. Figure 4.1(b) corresponds to a request that an employee has at least one weekend off during the planning horizon. Having one or more than one weekend off is associated with an acceptance level of 100. Having no weekend off is associated with an acceptance level of 60. The acceptance levels express the relative severity of the corresponding deviations. The lower the acceptance level, the more severe is the corresponding deviation. Hence, having no weekend off is more severe for an employee with a mapping function as the one shown in Figure 4.1(a), than for an employee with a mapping function as the one shown in Figure 4.1(b). Due to the lexicographic nature of the acceptance levels, the difference in severity is infinite and can thus not be quantified.

The SAP-LAL consists of finding an assignment of employees to work shifts such that all critical requests are accepted and that the number of deviations from the target values of noncritical requests is minimized. Thereby, a reduction in the number of less-accepted deviations is always preferred to any number of reductions in more-accepted deviations. For example, a schedule with four deviations associated with acceptance level 60 is always preferred to a schedule with only one deviation with acceptance level 50. The specific structure of the objective function requires the development of novel types of exact and heuristic solution approaches, which makes the SAP-LAL an interesting problem from the academic point of view. The problem is also interesting from the practical point of view, as the acceptance levels allow users to consider multiple conflicting requests in an

intuitive and direct manner. Due to the lexicographic ordering of the acceptance levels, the users have a clear understanding of how a change in the specification of acceptance levels affects the schedule.

4.2.2 Illustrative example

The planning horizon of the illustrative example spans two weeks. There are five different types of shifts: A, B, C, E (early shift), and L (late shift). Figure 4.2 shows for each shift type the start and end times, the set of employees that possess the required skills (compatible employees), and the days on which the corresponding shifts must be performed. Five employees (Ann, Bob, Dan, Eva, and Gil) can be assigned to the shifts subject to the types of critical and noncritical requests provided in Table 4.1. There are two requests of type 4: employee Bob wants to have Thursday and Friday of week 2 off. Refusing either of those two requests is associated with acceptance level 30. Table 4.2 lists all requests of the illustrative example. In total, there are 355 critical and noncritical requests, which we label with a number from 1 to 355. Some requests are given for each employee and day of the planning horizon. For example, there are 70 requests of type 1 because there are 5 employees and 14 days. Column 2 of Table 4.2 lists for each type the labels of the corresponding requests. Notice that the requests of type 2 and 3 affect shifts and not employees. Table 4.2 also contains the acceptance levels that are associated with refusing a request. Figures 4.1, 4.3, and 4.4 visualize the mapping functions of request types 9 to 13.

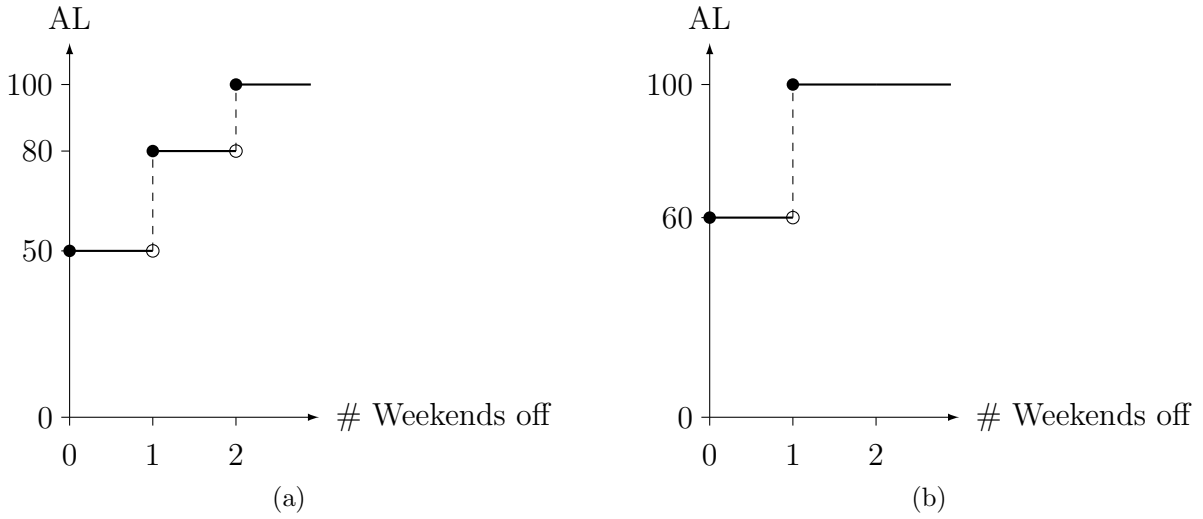


Figure 4.1: Mapping functions for requests of type 9

Shift		**Compatible**	**Week 1**							**Week 2**						
Start	**End**	**employees**	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
8am	4pm	Ann, Dan, Eva	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}
11am	7pm	Ann, Bob, Dan	B_1		B_3		B_5	B_6	B_7		B_9		B_{11}		B_{13}	B_{14}
4am	12pm	Dan, Eva, Gil	E_1	E_2	E_3	E_4	E_5			E_8	E_9	E_{10}	E_{11}	E_{12}		
3pm	11pm	Bob, Dan, Gil		L_2		L_4				L_8		L_{10}		L_{12}		
9am	5pm	Ann, Bob			C_3						C_9		C_{11}			

Figure 4.2: Illustrative example: shifts that need to be performed

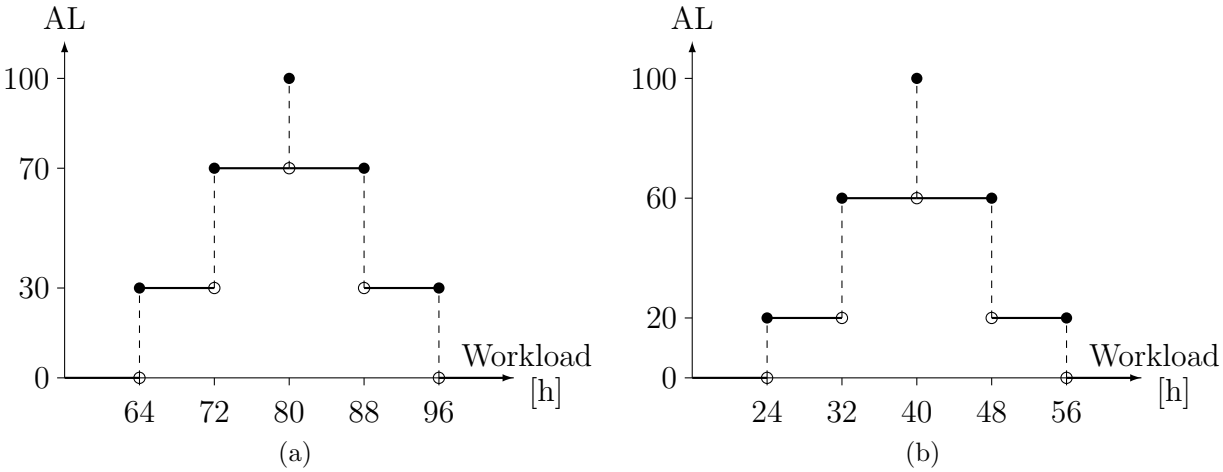


Figure 4.3: Mapping functions of request types 10 and 11

4.3 Literature review

In this section, we review existing solution techniques for employee scheduling problems. These techniques can be broadly divided into the three groups: mathematical programming-based techniques, metaheuristics, and matheuristics. Sections 4.3.1–4.3.3 contain a description of popular techniques from each group and discuss the difficulties that arise when applying these techniques to the SAP-LAL.

4.3.1 Mathematical programming-based techniques

Mathematical programming-based techniques appear to be the most popular ones for employee scheduling problems (cf. Van den Bergh et al., 2013). These approaches model the employee scheduling problem as a linear, integer or mixed-integer program that is solved either with a general-purpose solver or a specific algorithm such as column gener-

Table 4.2: Requests for illustrative example

Type	Request	Affected employees	AL
1	1 – 70	Ann, Bob, Dan, Eva, Gil	0
2	71 – 111	—	0
3	112 – 152	—	0
4	153 – 154	Bob (wants days 11, 12 off)	30
5	155 – 199	Ann, Bob, Dan, Eva, Gil	60
6	200 – 259	Ann, Bob, Dan, Eva, Gil	60
7	260 – 324	Ann, Bob, Dan, Eva, Gil	1
8	325 – 334	Ann, Bob, Dan, Eva, Gil	30
9	335 – 336	Ann, Dan	Fig. 4.1(a)
	337 – 339	Bob, Eva, Gil	Fig. 4.1(b)
10	340 – 342	Ann, Bob, Dan	Fig. 4.3(a)
	343 – 344	Eva, Gil	Fig. 4.3(b)
11	345 – 347	Ann, Bob, Dan	Fig. 4.3(a)
	348 – 349	Eva, Gil	Fig. 4.3(b)
12	350 – 352	Dan, Eva, Gil	Fig. 4.4(a)
13	353 – 355	Bob, Dan, Gil	Fig. 4.4(b)

ation, branch-and-price, or Lagrangian relaxation. The main advantage of mathematical programming-based techniques is the flexibility to accommodate a large variety of requests in the underlying mathematical programming formulation.

For problems with multiple conflicting requests, the literature on mathematical programming-based techniques concentrates on goal programming formulations (cf., e.g., Jones and Tamiz, 2002, 2010; Romero, 2014; Jones and Tamiz, 2016). In goal programming, each request is associated with a target value, and deviations from target values are captured by deviational variables. A so-called achievement function penalizes the deviations according to the preferences of the decision maker. The main variants of goal programming are lexicographic, Chebyshev and weighted goal programming.

Lexicographic goal programming requires a ranking of the requests that reflects their importance. The unwanted deviations from the target values are minimized sequentially according to the given ranking. This variant has been used by decision makers who do not need to model trade-offs between requests because they have a clear ranking of the requests in mind (cf., e.g., Berrada et al., 1996). Chebyshev goal programming minimizes

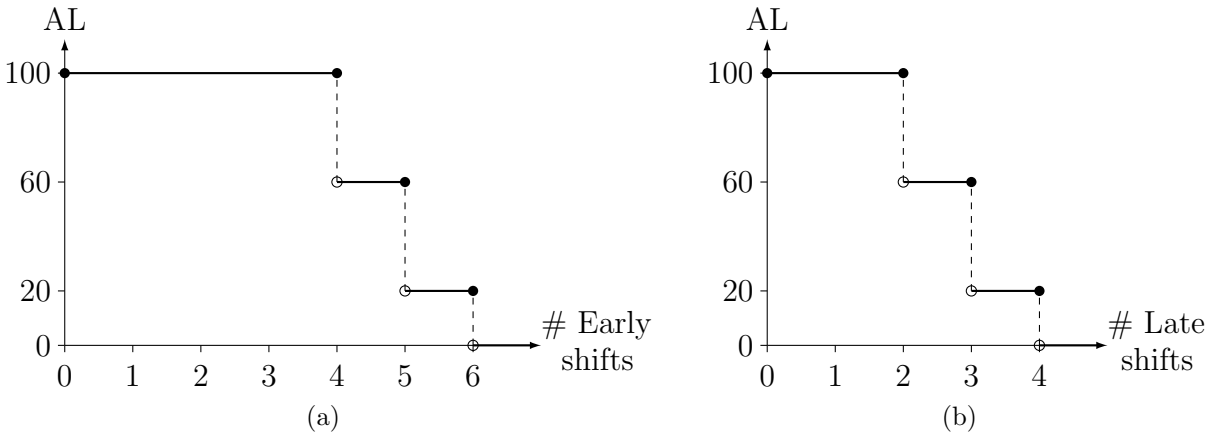


Figure 4.4: Mapping functions of request types 12 and 13

the maximum unwanted deviation across all requests. It has been used by decision makers who are interested in accepting requests in a balanced manner (cf., e.g., Ignizio, 2004).

Weighted goal programming allows for direct trade-offs among requests. A weight is defined for each deviational variable that quantifies its relative importance. The achievement function is the weighted sum of the deviational variables. The traditional form of weighted goal programming assumes that the weights are constant and do not change at further distances from the target value (cf., e.g., Beaulieu et al., 2000). As this assumption is too restrictive to fit the preferences of many decision makers, various extensions have been proposed.

Charnes and Collomb (1972) introduced interval goal programming, which allows decision makers to specify a target interval instead of a target value. Deviations from either end of the interval are penalized in the achievement function. Subsequently, Charnes et al. (1976) introduced penalty functions that penalize large deviations by imposing a higher weight than that assigned to small deviations. This idea was extended by Kvanli (1980) and Jones and Tamiz (1995), who proposed more complex penalty functions including decreasing functions, functions with discontinuities and non-linear functions. Romero (2004) consolidated U-shaped penalty functions in an achievement function with a general structure. This achievement function also encompasses the basic variants of lexicographic and Chebyshev goal programming. The use of complex penalty functions requires the introduction of binary variables and additional constraints, which increases the computational cost. To address this drawback of interval goal programming models, Chang (2006) and Chang and Lin (2009) proposed techniques to reduce the number of variables and constraints required to model specific penalty functions.

A shared drawback of mathematical programming-based techniques is that the un-

derlying models contain a substantial number of constraints and variables when they are formulated for large-sized instances. Despite the recent improvements in optimization software and computer hardware (cf., e.g., Lodi, 2010; Koch et al., 2011; Bixby, 2012), it is often the case for large instances that these techniques do not return any feasible solution in a reasonable amount of computation time. Furthermore, specific difficulties arise for individual groups of techniques. The existing lexicographic goal programming approaches fail to consider trade-offs between requests as they optimize the requests sequentially. For example, assume that for a problem instance with only one employee and two types of requests only three feasible schedules (A, B, and C) exist. The first request is to have a workload of at most 30 hours and the second request is to have four weekends off. In schedule A, the employee has four weekends off, but exceeds the target workload by 20 hours. In schedule B, the employee has only one weekend off, but the target workload request is met. In schedule C, the employee has three weekends off and the workload exceeds the target by only 1 hour. Existing lexicographic goal programming approaches would always select schedule A or B, but never schedule C, although this appears to be the most favourable schedule. The existing weighted goal programming approaches and their extensions allow for a more accurate modelling of the decision maker's preferences and have been applied to many real-world applications (cf. the reviews of Tamiz et al., 1995; Jones and Tamiz, 2002). However, to apply weighted goal programming to the SAP-LAL, the penalty functions need to assign weights in such a way that the weight of a less-accepted deviation is always greater than the sum of all weights of more-accepted deviations. For example, for a small problem instance with 40 different acceptance levels and ten deviational variables per acceptance level, the largest weight has to be more than 10^{40} times larger than the smallest weight. Such large numbers may slow down commercial solvers or even cause numerical problems. The same difficulties arise for Chebyshev goal programming approaches where the maximum weighted deviation across all requests is minimized. Therefore, in the Section 4.4, we present an exact approach based on lexicographic goal programming that unlike existing lexicographic goal programming approaches is able to consider trade-offs among requests.

4.3.2 Metaheuristics

An important group of approaches for employee scheduling problems are metaheuristics. This type of solution approach has been successfully used for real-world problems where exact approaches are not able to devise satisfactory solutions within an acceptable time limit. The general idea is to iteratively improve a single solution or a population of solutions with a local improvement procedure until a stopping criterion is met. In addition

to a local improvement procedure, metaheuristics also employ various search strategies to escape from local optima. Metaheuristics tend to find good solutions in a reasonable amount of computation time when a) it is easy to construct a feasible solution quickly, b) the solution space is smooth, i.e., promising search directions can be determined easily, and c) when a direct representation of a solution exists. The most popular metaheuristics for employee scheduling problems are simulated annealing (cf., e.g., Bertels and Fahle, 2006; Cordeau et al., 2010; Smet et al., 2014a), tabu search (cf., e.g., Dowsland, 1998; Bard and Wan, 2006; Bester et al., 2007), and genetic algorithms (cf., e.g., Aickelin and Dowsland, 2000, 2004; Maenhout and Vanhoucke, 2008; Valls et al., 2009; Bai et al., 2010).

Although the main structure of metaheuristics is generic, a problem-specific implementation and fine tuning of parameters is necessary to obtain satisfactory performance (cf., e.g., Kopanos et al., 2010). This complicates the adaption of the solution approach to small changes in the problem setting as for example additional requests. In contrast to exact approaches, a further disadvantage of metaheuristics is that they cannot systematically evaluate the quality of the generated solutions. Regarding the application of metaheuristics to the SAP-LAL, the main difficulty is that due to the lexicographic nature of acceptance levels, the solution space of typical problem instances is non-smooth, which makes it difficult to find promising search directions. Furthermore, the large number of conflicting requests reduces the effectiveness of metaheuristics in general (cf., e.g., Jones et al., 2002) as evaluating the quality of a solution requires more computation time.

4.3.3 Matheuristics

Recently, matheuristics have delivered promising results for various scheduling problems. Raidl and Puchinger (2008), Boschetti et al. (2009), Maniezzo et al. (2009) and Ball (2011) provide general reviews of matheuristics. Matheuristics combine the flexibility of mathematical programs to easily accommodate complex constraints and the ability of heuristics to find good solutions quickly. The basic idea is to employ the mathematical program for solving specific subproblems of the original problem for which metaheuristics have difficulties in dealing with. The size of the subproblems can be adjusted to ensure fast and predictable optimization behavior (cf. Kopanos et al., 2010). A stable optimization behaviour is particularly important in a dynamic setting, i.e., when requests are frequently modified or new requests have to be considered.

In the context of employee scheduling, only few matheuristics have been introduced. These matheuristics belong either to the group of constructive matheuristics or to the group of improvement matheuristics. The former iteratively generates a feasible solution, whereas the latter takes as input an initial solution which is improved iteratively.

Smet et al. (2014b) propose a constructive matheuristic for the shift minimization personnel task scheduling problem. The solution is constructed by iteratively assigning subsets of employees to tasks using an integer program until all tasks have been assigned. The subsets of employees are selected randomly without using any information of the current solution.

Smet and Vanden Berghe (2012) propose an improvement matheuristic for the problem considered in Smet et al. (2014b). In each improvement iteration randomly selected subsets of employees are reassigned to tasks. For the same problem, Smet et al. (2014b) use the concept of local branching (cf. Fischetti and Lodi, 2003) to define the subproblems such that only a limited number of binary variables can change their values.

Della Croce and Salassa (2014) provide an exact formulation and an improvement matheuristic for a nurse rostering problem that stems from an Italian hospital. To obtain an initial solution, the exact formulation is solved for a prescribed time limit. This solution is then iteratively improved by the matheuristic. In each iteration, only a small number of decision variables are allowed to change their values. This is achieved by either using the concept of local branching or imposing lower and upper bounds directly on the variables.

The performance of matheuristics depends strongly on the definition of the subproblems. Ideally, the subproblems are defined such that a) the complexity of the subproblem makes a mathematical program the most appropriate solution methodology, b) the subproblem focuses only on the critical decisions that have a direct impact on the quality of the solution, and c) the corresponding mathematical program can be formulated in a compact way without redundant constraints and variables. The strategies of the above described matheuristics to define the subproblems can in principle be applied to the SAP-LAL. However, these strategies do not appear to be suitable for the following reasons. The improvement matheuristic of Della Croce and Salassa (2014) constructs the subproblems by imposing additional constraints on the decision variables, either by using the concept of local branching or by imposing lower and upper bounds on variables to fix their values. A major drawback of local branching is that the size of the subproblem in terms of number of decision variables and constraints is equally big as the original full problem. As the full formulation for typical instances of the SAP-LAL is already large, imposing additional constraints leads to subproblem formulations that are difficult to handle. Fixing variables by imposing upper and lower bounds has a similar disadvantage as the size of the subproblem formulations in terms of number of decision variables and constraints can only exceed the size of the full formulation. Although the preprocessing procedures of state-of-the-art solvers are able to remove most of the fixed variables and the corresponding constraints, reading such large models in each iteration considerably decreases the performance of the

overall approach. Also the matheuristic of Smet et al. (2014b) relies on local branching and is thus not appropriate for the SAP-LAL. The matheuristic of Smet and Vanden Berghe (2012) defines subproblems by randomly selecting a set of employees. Selecting employees randomly often creates subproblems, which do not consider the critical decisions that have a direct impact on the solution quality. In Section 4.6, we demonstrate that selecting employees specifically is indeed superior to selecting employees randomly.

4.4 Exact solution approach

In Rihm and Baumann (2015b), we introduced a preliminary version of the exact solution approach. In this paper, we extend the preliminary version by including all requests related to early and late shifts. Furthermore, we developed a simplified notation and presentation of the lexicographic goal program, and we provide a more detailed explanation of the constraints.

The exact solution approach consists of two phases. In the first phase, each request is decomposed into a set of sub-requests (see Section 4.4.1). In the second phase, a lexicographic goal program is formulated to iteratively optimize the sub-requests (see Section 4.4.2). In Section 4.4.3, we illustrate the exact solution approach by means of the illustrative example that we introduced in Section 4.2.2.

4.4.1 Phase 1: Request decomposition

The goal of the decomposition is to transform the original problem into a problem that can be solved efficiently with lexicographic goal programming. Lexicographic goal programs require a clear order of the goals to be optimized. Such an order cannot be found for the requests directly because a single request might be associated with different acceptance levels. We therefore decompose the requests into so-called sub-requests that are associated with only one acceptance level. It follows that each sub-request can either be accepted or refused. The sub-requests can then be sorted in ascending order of their acceptance levels. The decomposition is achieved as follows. The number of sub-requests is equal to the number of kinks in the mapping function of the original request. Each sub-request is assigned the target value and the acceptance level from the corresponding kink in the mapping function.

Figure 4.5 illustrates the decomposition of the workload request of Figure 4.3(a). This request can be refused to different degrees, and is therefore decomposed into four sub-requests. The first sub-request (bottom left plot in Figure 4.5) is refused when the workload is less than 72 hours. Such a refusal is associated with an acceptance level of 30.

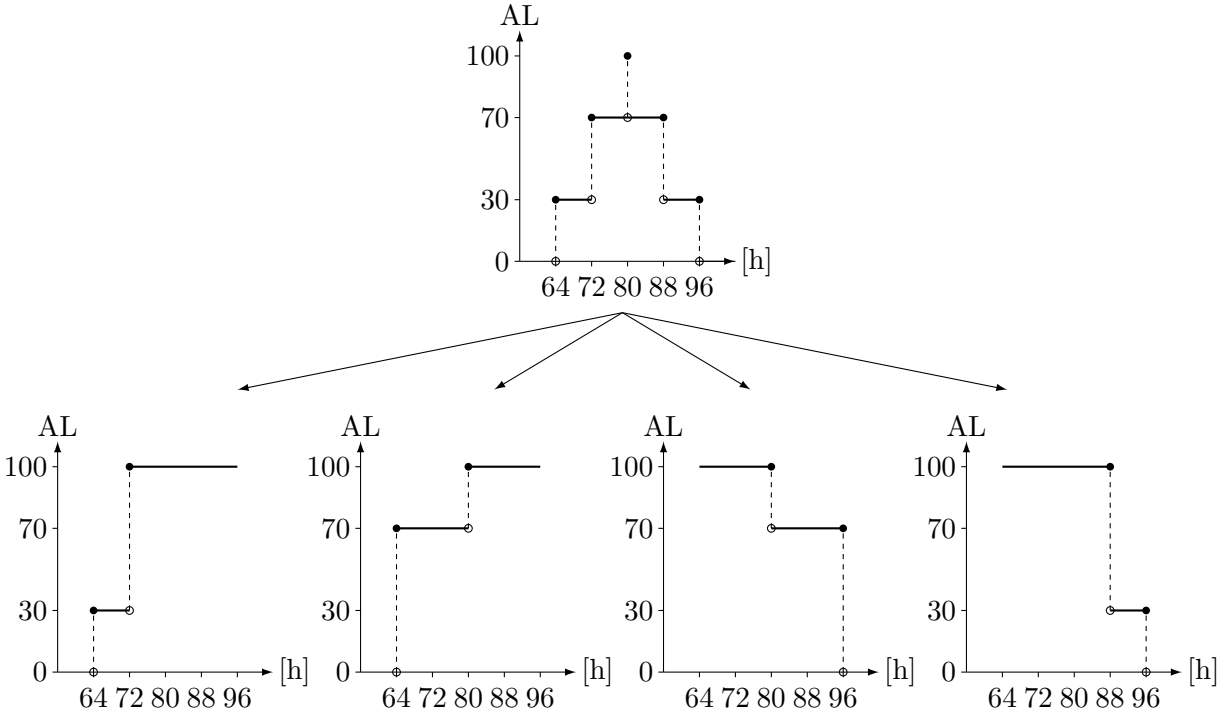


Figure 4.5: Decomposition of a request into four sub-requests.

The second sub-request (second plot from the left) is refused when the workload is less than 80 hours. Such a refusal is associated with an acceptance level of 70. The meaning of the third and fourth sub-request can be described analogously. A workload of less than 64 hours, or of more than 96 hours is infeasible. After the decomposition, each of the four sub-requests is associated to exactly one acceptance level, so a clear order of the sub-requests results.

4.4.2 Phase 2: Lexicographic goal program

In the second phase, we formulate and solve a lexicographic goal program (LGP). For each sub-request, we introduce a binary deviational variable that is equal to one when the sub-request is refused. The LGP is solved as a series of binary linear programs (BLPs). The first BLP minimizes the number of refused sub-requests associated with the lowest acceptance level, the second BLP minimizes the number of refused sub-requests associated with the second-lowest acceptance level, etc. Prior to solving the next BLP, an additional constraint is added to ensure that the number of refusals from the previous optimization will not be exceeded in subsequent optimizations. Note that the additional constraints do not fix assignments because the corresponding deviational variables can still change in subsequent optimizations as long as the total number of refusals does not exceed the

prescribed upper bound.

A distinctive feature of our approach is that all deviational variables are binary, which provides great flexibility for extensions; e.g., a balanced distribution of refused sub-requests among employees could be easily incorporated. In Rihm and Baumann (2015a), we introduced an approach for improving an existing schedule in terms of fairness that takes advantage of this feature.

We introduce the notation in Section 4.4.2.1, formulate the lexicographic goal program in Section 4.4.2.2 and present aggregation techniques for reducing the number of constraints in Section 4.4.2.3.

4.4.2.1 Notation

We use the following notation.

Indices

a	Acceptance level
d	Day
i	Employee
q	Request type
r	Sub-request
s	Shift
w	Weekend

Sets

A Acceptance levels

D Days

D_w Days of weekend w

I Employees

I_s Employees compatible with shift s

R_a Sub-requests with acceptance level a

R^q Sub-requests of type q

R_i^q Sub-requests of type q of employee i

S Shifts

S_{id} Compatible shifts of employee i starting on day d

S_{id}^E Compatible early shifts of employee i starting on day d

S_{id}^L Compatible late shifts of employee i starting on day d

S_{id}^b Pairs of shifts between which the rest period (break) is less than b hours

W Weekends

Parameters

d_r Day relevant for sub-request r

g_r Coefficient of deviational variable z_r for the basic formulation (BF)

h_r Coefficient of deviational variable z_r for the aggregated formulation (AF)

i_r Employee relevant for sub-request r

l_s Length of shift s

t_r Target value of sub-request r

v_a Allowed number of refused sub-requests at acceptance level a

w_r Weekend relevant for sub-request r

Variables

x_{is} $= 1$, if employee i is assigned to shift s; $= 0$, else

y_{iw} $= 1$, if employee i has weekend w off; $= 0$, else

z_r $= 1$, if sub-request r is refused; $= 0$, else

4.4.2.2 Model formulation

The model presented below covers the three types of critical requests and the ten types of noncritical requests presented in Table 4.1. In the following, we refer to sub-requests that stem from a noncritical request of type q as sub-requests of type q. The model is solved multiple times, once for each unique acceptance level $a^* \in A$, in ascending order of acceptance levels.

The objective function minimizes the total number of refused sub-requests associated with acceptance level a^*.

$$\text{Min} \quad \sum_{r \in R_{a^*}} z_r$$

Constraints (4.1) bound the number of refused sub-requests for all acceptance levels $a < a^*$ to ensure that the results from previous optimizations are preserved.

$$\sum_{r \in R_a} z_r \leq v_a \qquad (a \in A : \ a < a^*) \tag{4.1}$$

Constraints (4.2) address the requests of type 1. They ensure that each employee $i \in I$ is assigned to at most one shift each day $d \in D$.

$$\sum_{s \in S_{id}} x_{is} \leq 1 \qquad (i \in I, \ d \in D) \tag{4.2}$$

Constraints (4.3) address the requests of type 2. They ensure that exactly one employee is assigned to each shift. Notice that sets I_s and S_{id} are defined such that the critical requests of type 3 cannot be refused.

$$\sum_{i \in I_s} x_{is} = 1 \qquad (s \in S) \tag{4.3}$$

Constraint (4.4) is formulated for each sub-request of type 4. Sub-requests of type 4 represent preferences for days off and are specified for specific employees and days of the planning horizon. The target value t_r is zero and the coefficient g_r is one for all sub-requests $r \in R^4$. A sub-request $r \in R^4$ is refused when the left-hand side is equal to one, that is, when employee i_r is assigned to a shift on day d_r. In this case, variable z_r is forced

to take value one.

$$\sum_{s \in S_{i_r d_r}} x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^4) \tag{4.4}$$

Constraint (4.5) covers each sub-request of type 5. Sub-requests of type 5 are specified for all employees $i \in I$ and all days $d \in D$, where $d > t_r$. They are intended to prevent employees from being assigned to shifts on more than $t_r = 5$ consecutive days. Sub-request r is refused when employee i_r is assigned to a shift on day d_r in addition to shifts on days $d_r - 1, d_r - 2, \ldots, d_r - t_r$. The sub-requests for the first $d < t_r$ days of the planning horizon could easily be incorporated by including the last days of the previous planning period in the planning horizon and fixing the corresponding decision variables.

$$\sum_{d'=d_r-t_r}^{d_r} \sum_{s \in S_{i_r d'}} x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^5) \tag{4.5}$$

Constraint (4.6) is formulated for each sub-request of type 6. Sub-requests of type 6 are specified for all employees $i \in I$ and all days $d \in D$, where $1 < d < |D|$. They assume that employees prefer two consecutive days off. A sub-request r is refused when employee i_r has a day off on day d_r and is assigned to a shift on day $d_r - 1$ and a shift on day $d_r + 1$. The target value t_r and the coefficient g_r are equal to one for all sub-requests $r \in R^6$.

$$\sum_{s \in S_{i_r d_r - 1}} x_{i_r s} - \sum_{s \in S_{i_r d_r}} x_{i_r s} + \sum_{s \in S_{i_r d_r + 1}} x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^6) \tag{4.6}$$

Constraints (4.7) cover all sub-requests of type 7. They are intended to provide employees a b-hour rest period between consecutive shifts. There is one sub-request r for each employee i and day $d \geq 2$. Set S^b_{id} contains all pairs of shifts $(s_1 \in S_{id-1}, s_2 \in S_{id})$ between which the period off is less than b hours long. The request is refused if both of these shifts are assigned to the same employee, that means if the left-hand-side is equal to two. The target value t_r and the coefficient g_r are equal to one for all sub-requests $r \in R^7$.

$$x_{i_r s_1} + x_{i_r s_2} \leq t_r + g_r z_r \qquad (r \in R^7, (s_1, s_2) \in S^b_{i_r d_r}) \tag{4.7}$$

Constraints (4.8) address all sub-requests of type 8, which are intended to assign employees either no weekend shifts or one shift on each day of the weekend. The binary variable y_{iw} indicates whether employee $i \in I$ has weekend $w = (d_1, d_2) \in W$ off. Variables y_{iw} are reused to model the sub-requests of type 9. The target value is $t_r = 2$ and the coefficient

is $g_r = -1$ for all sub-requests $r \in R^8$.

$$2y_{i_r w_r} + \sum_{d \in D_{w_r}} \sum_{s \in S_{i_r d}} x_{i_r s} = t_r + g_r z_r \qquad (r \in R^8) \tag{4.8}$$

Constraints (4.9) cover sub-requests of type 9 and impose a minimum number of weekends off per employee.

$$\sum_{w \in W} y_{i_r w} \geq t_r + g_r z_r \qquad (r \in R^9) \tag{4.9}$$

Constraints (4.10) cover sub-requests of type 10, which are intended to ensure that the target workloads of employees are not exceeded.

$$\sum_{d \in D} \sum_{s \in S_{i_r d}} l_s x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^{10}) \tag{4.10}$$

The left-hand side computes the total workload of employee i_r by summing over all days $d \in D$ and shifts $s \in S_{i_r d}$ planned on that day. If this workload exceeds the target value t_r, variable z_r is forced to take value one, which corresponds to a refusal of sub-request r. In this case, the right-hand side is equal to $t_r + g_r$, which is a hard upper bound for the workload. Analogously, constraints (4.11) cover request type 11, which is intended to prevent that the actual workload of employees falls below the respective target workloads.

$$\sum_{d \in D} \sum_{s \in S_{i_r t}} l_s x_{i s} \geq t_r + g_r z_r \qquad (r \in R^{11}) \tag{4.11}$$

Constraints (4.12) cover sub-requests of type 12, which are intended to ensure that an employee's target number of early shifts is not exceeded.

$$\sum_{d \in D} \sum_{s \in S^E_{i_r d}} x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^{12}) \tag{4.12}$$

Constraints (4.13) address sub-requests of type 13, which are intended to ensure that an employee's target number of late shifts is not exceeded.

$$\sum_{d \in D} \sum_{s \in S^L_{i_r d}} x_{i_r s} \leq t_r + g_r z_r \qquad (r \in R^{13}) \tag{4.13}$$

Finally, all variables are binary.

$$x_{is} \in \{0,1\} \qquad (i \in I_s,\ s \in S) \qquad (4.14)$$

$$y_{iw} \in \{0,1\} \qquad (i \in I,\ w \in W) \qquad (4.15)$$

$$z_r \in \{0,1\} \qquad (r \in R) \qquad (4.16)$$

4.4.2.3 Model size reduction

The size of the formulation in terms of constraints can be reduced using the following techniques. A first reduction can be achieved by aggregating sub-requests of the same type and of the same employee. More precisely, each request with two or more kinks in the mapping function can be described by only one constraint per employee. Thereby, the deviational variables z_r denote to which degree the request is refused. We need to define set R_i^q containing all sub-requests of type q relevant for employee i. We introduce parameter h_r, which captures the distance between one kink and its adjacent kink with a lower acceptance level, i.e.,

$$h_r = \begin{cases} g_r - \max_{r' \in R_i^q : g_r > g_{r'}}(g_{r'}), & \text{if } g_r \geq 0; \\ g_r - \min_{r' \in R_i^q : g_r < g_{r'}}(g_{r'}), & \text{otherwise.} \end{cases}$$

Constraints (4.17) aggregate constraints (4.9).

$$\sum_{w \in W} y_{i_r w} \geq \max_{r \in R_i^9}(t_r) + \sum_{r \in R_i^9} h_r z_r \qquad (i \in I) \qquad (4.17)$$

Analogously, we aggregate constraints (4.10), (4.11), (4.12) and (4.13):

$$\sum_{d \in D} \sum_{s \in S_{i_r d}} l_s x_{i_r s} \leq \min_{r \in R_i^{10}}(t_r) + \sum_{r \in R_i^{10}} h_r z_r \qquad (i \in I) \qquad (4.18)$$

$$\sum_{d \in D} \sum_{s \in S_{i_r d}} l_s x_{i_r s} \geq \max_{r \in R_i^{11}}(t_r) + \sum_{r \in R_i^{11}} h_r z_r \qquad (i \in I) \qquad (4.19)$$

$$\sum_{d \in D} \sum_{s \in S^E_{i_r d}} x_{i_r s} \leq \min_{r \in R_i^{12}}(t_r) + \sum_{r \in R_i^{12}} h_r z_r \qquad (i \in I) \qquad (4.20)$$

$$\sum_{d \in D} \sum_{s \in S^L_{i_r d}} x_{i_r s} \leq \min_{r \in R_i^{13}}(t_r) + \sum_{r \in R_i^{13}} h_r z_r \qquad (i \in I) \qquad (4.21)$$

The model size can further be reduced by aggregating sub-requests of different types. This is possible when two requests are structurally similar (i.e. identical left-hand side), affect

the same employee and share the same target value. Here, this only applies to request types 10 and 11.

$$\sum_{d\in D}\sum_{s\in S_{id}} l_s x_{is} = \min_{r\in R_i^{10}}(t_r) + \sum_{r\in R_i^{10}\cup R_i^{11}} h_r z_r \qquad (i \in I) \qquad (4.22)$$

For this last aggregation, $\min_{r\in R_i^{10}}(t_r) = \max_{r\in R_i^{11}}(t_r)$ must apply.

In the experimental study in Section 4.7, we compare the performance of the basic model formulation (BF) of Section 4.4.2.2 with the performance of the aggregated model formulation (AF) of Section 4.4.2.3:

$$(BF)\ \begin{cases} \text{Min} & \displaystyle\sum_{r\in R_{a^*}} z_r \\ \text{s.t.} & (4.1)-(4.16) \end{cases} \qquad (AF)\ \begin{cases} \text{Min} & \displaystyle\sum_{r\in R_{a^*}} z_r \\ \text{s.t.} & (4.1)-(4.8) \\ & (4.14)-(4.17) \\ & (4.20)-(4.22) \end{cases}$$

4.4.3 Illustrative example

In this section, we apply the exact approach to the illustrative example introduced in Section 4.2.2. Table 4.3 reports the result of the decomposition phase. The requests have been decomposed into 373 sub-requests. The fourth column of Table 4.3 contains the domain of each sub-request, the employee and, if existing, the exact day to which the corresponding sub-request applies. The acceptance levels and target values associated with the sub-requests are listed in the fifth and sixth columns of Table 4.3. The last column specifies the maximum positive or negative undesired deviation from the target value that is still considered feasible.

Sub-request 153 applies to employee Bob on day 11. If Bob has to work on this day, a sub-request associated with acceptance level 30 is refused.

The information given for sub-requests 335–338 is as follows. Ann and Dan ideally have two or more weekends off, which is expressed by the target value of sub-requests 335 (Ann) and 336 (Dan), respectively. From $g_r = -2$, it follows that the hard lower bound on the number of weekends off is 0. Having only one weekend off complies with sub-request 337 (338) but leads to a refusal of sub-request 335 (336). Such a refusal is associated with an acceptance level of 80. Having no weekend off additionally leads to a refusal of sub-request 337 (338), which is associated with an acceptance level of 50.

Figure 4.6 depicts an optimal schedule for the illustrative example. All refused sub-

Table 4.3: Sub-requests for the illustrative example

Type q	Request	Sub-request r	Domain	AL a	Target value t_r	Parameter g_r
1	1–70	1–70	$i_r \in I, d_r \in D$	0	—	—
2	71–111	71–111	$s \in S$	0	—	—
3	112–152	112–152	$s \in S$	0	—	—
4	153	153	$i_r =$Bob, $d_r = 11$	30	0	1
	154	154	$i_r =$Bob, $d_r = 12$	30	0	1
5	155–163	155–163	$i_r =$Ann, $d_r = 6, \ldots, \lvert D \rvert$	60	5	1
	164–172	164–172	$i_r =$Bob, $d_r = 6, \ldots, \lvert D \rvert$	60	5	1
	173–199	173–199	$i_r =$Dan,Eva,Gil, $d_r = 6, \ldots, \lvert D \rvert$	60	5	1
6	200–259	200–259	$i_r \in I, d_r \in D : 2 \leq d_r \leq \lvert D \rvert - 1$	60	1	1
7	260–324	260–324	$i_r \in I, d_r \in D : d_r \geq 2$	1	1	1
8	325–334	325–334	$i_r \in I, w_r \in W$	30	2	-1
9	335–336	335–336	$i_r \in$ {Ann,Dan}	80	2	-2
		337–338	$i_r \in$ {Ann,Dan}	50	1	-1
	337–339	339–341	$i_r \in$ {Bob,Eva,Gil}	60	1	-1
10	340–342	342–344	$i_r \in$ {Ann,Bob,Dan}	70	80	16
		345–347	$i_r \in$ {Ann,Bob,Dan}	30	88	8
	343–344	348–349	$i_r \in$ {Eva,Gil}	60	40	16
		350–351	$i_r \in$ {Eva,Gil}	20	48	8
11	345–347	352–354	$i_r \in$ {Ann,Bob,Dan}	70	80	-16
		355–357	$i_r \in$ {Ann,Bob,Dan}	30	72	-8
	348–349	358–359	$i_r \in$ {Eva,Gil}	60	40	-16
		360–361	$i_r \in$ {Eva,Gil}	20	32	-8
12	350–352	362–364	$i_r \in$ {Dan,Eva,Gil}	60	4	2
		365–367	$i_r \in$ {Dan,Eva,Gil}	20	5	1
13	353–355	368–370	$i_r \in$ {Bob,Dan,Gil}	60	2	2
		371–373	$i_r \in$ {Bob,Dan,Gil}	20	3	1

	Week 1							Week 2						
Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Ann	A_1	A_2	C_3	A_4	A_5				C_9	A_{10}	C_{11}	A_{12}	A_{13}	A_{14}
Bob	B_1	L_2	B_3		B_5	B_6	B_7	L_8	B_9				B_{13}	B_{14}
Dan	E_1	E_2	A_3	E_4	E_5			A_8	A_9	L_{10}	B_{11}	L_{12}		
Eva			E_3			A_6	A_7	E_8			A_{11}			
Gil				L_4					E_9	E_{10}	E_{11}	E_{12}		

Figure 4.6: Optimal schedule

Table 4.4: Refused sub-requests

AL a	Request type t	Sub-request r	Affected employee i_r
60	5	163	Ann ($d_r = 14$)
60	6	214	Bob ($d_r = 4$)
60	9	339	Bob
70	10	342	Ann
80	9	335	Ann

requests are listed in Table 4.4 in ascending order of their corresponding acceptance levels.

Both model formulations (BF and AF) lead to the same schedule, and the corresponding CPU times are negligible ($\ll 1$ s).

4.5 Matheuristic

Despite improvements in optimization software and computer hardware, exact approaches are only applicable to small- and medium-sized problem instances. For large-sized instances, heuristic solution procedures are required. According to Cordeau et al. (2002), good heuristics are not only accurate and fast but also simple and flexible. We designed a matheuristic for the SAP-LAL because a) matheuristics have proven to deliver high-quality solutions for related problems (cf., e.g., Smet and Vanden Berghe, 2012; Della Croce and Salassa, 2014; Smet et al., 2014b), b) the speed can be controlled by the

size of the subproblems, c) when using an algebraic modeling language the implementation effort is rather low, and d) the underlying BLP model offers flexibility to account for additional request types. In Section 4.5.1, we describe the matheuristic in detail. In Section 4.5.2, we apply the matheuristic to the illustrative example that we introduced in Section 4.2.2.

4.5.1 Matheuristic: description

The basic idea of the matheuristic is to iteratively improve an initial solution by reassigning groups of employees (see Figure 4.7). In the following, parameter k denotes the size of such groups of employees. Parameter k controls the size of the subproblems and thereby the degree of optimization in the matheuristic. A small value of k leads to small subproblems which can be solved in short CPU time. However, the corresponding improvements tend to be incremental. Larger improvements can be obtained for larger values of k, but at the cost of increased computational effort. In our experiments, we select parameter k independently of the problem size, which leads to subproblems of similar size for all problem instances. This has the advantage that for all instances the size of the subproblems is comparable and thus the optimization behaviour of the matheuristic is barely affected by the problem size. For the construction of the initial solution and the improvement iterations we use model (AF). The initial solution is constructed by solving model (AF) without noncritical requests. The resulting solution is feasible because it complies with all critical requests. Then, an improvement routine is executed for each acceptance level in ascending order of acceptance levels. The improvement routine is executed multiple times for the same acceptance level until one of the following three stopping criteria is satisfied: a) the current solution does not contain refused sub-requests that are associated with the current acceptance level (REF criterion), b) no improvement was achieved for a predefined number of subproblems (IMP criterion) and c) a predefined CPU time limit has been reached for one acceptance level (CPU criterion). We refer to this time limit as acceptance level time limit. The time limit imposed on the individual subproblems is hereinafter referred to as subproblem time limit.

For a given acceptance level a^*, the improvement routine performs the following three steps:

1. A subset of k employees is selected according to a selection rule.

2. Model (AF) is formulated for the selected employees and acceptance level a^*. The resulting model is very compact, as it contains only variables and constraints related to the selected employees.

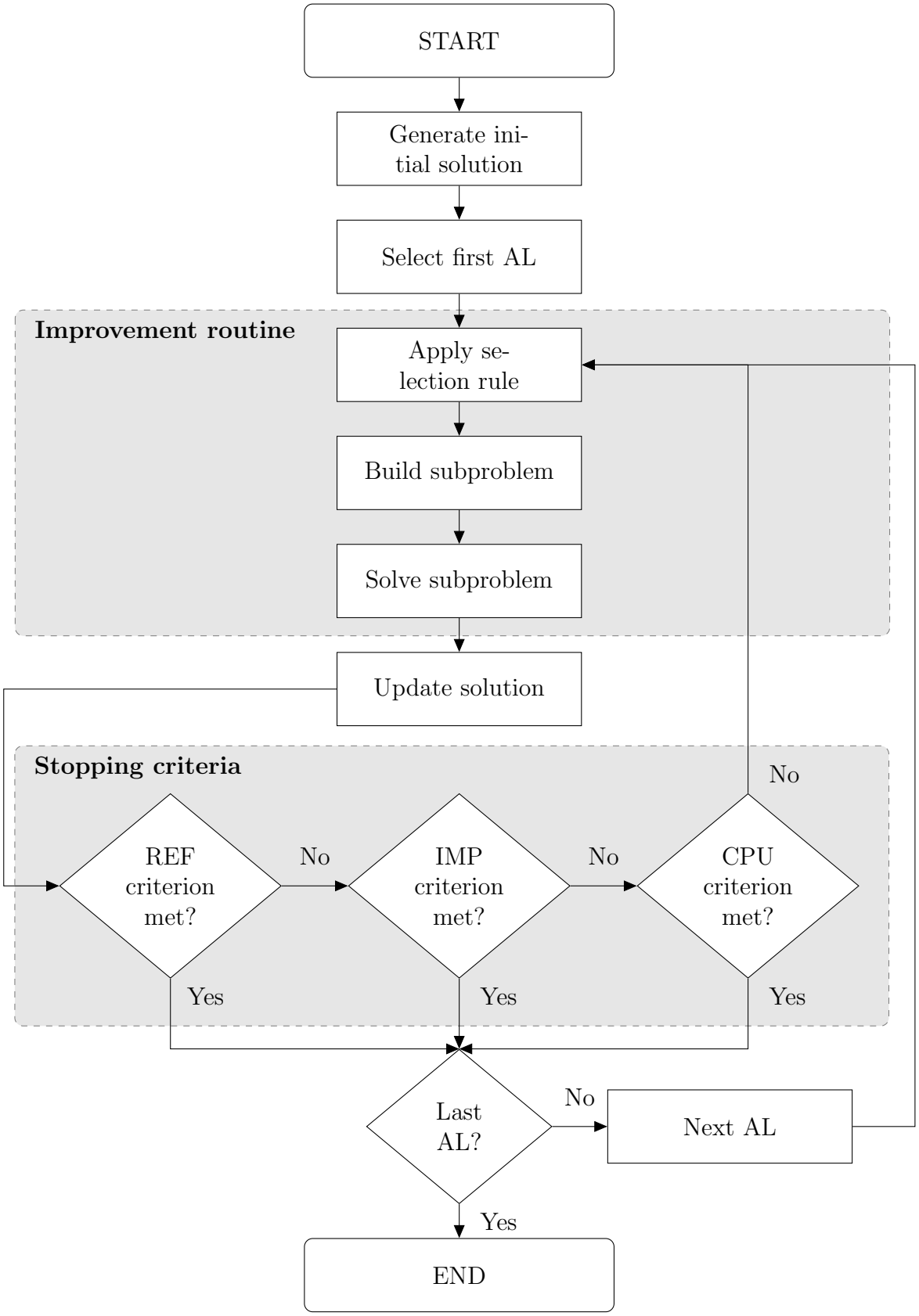


Figure 4.7: Flowchart of matheuristic

3. The reduced model is solved.

The k employees could be selected randomly without considering any property of the current solution. However, randomly selected groups of employees might not have any refused sub-requests associated with acceptance level a^*. We therefore employ the following selection rule:

1. Among all employees who have at least one refused sub-request associated with acceptance level a^*, one employee is selected randomly. The employee is denoted by i^*.

2. Among all refused sub-requests with acceptance level a^* of employee i^*, one sub-request is randomly selected. This selected sub-request is denoted by r^*.

3. Among all employees who could prevent the refusal of sub-request r^* by swapping one of their shifts with employee i^*, one employee is selected. Thereby, we distinguish two cases:

   - Case 1: Sub-request r^* is of types 4–8: The refusal occurs because a shift s is assigned to employee i^* on a particular day d. We select one employee having that day off and the required skills to perform shift s. We should note that this employee could prevent the refusal, but it is not ensured that it is not at the expense of a new refusal.
   - Case 2: Sub-request r^* is of types 9–13: The refusal occurs because too many/few shifts are assigned to employee i^* over the entire planning period. We select one employee with the same skills.

4. We randomly select $k - 2$ other employees.

The idea of constructing subproblems is to reduce the search space so that a general-purpose solver can solve them quickly and a large number of iterations can be performed in a short amount of time. Another advantage is that the solver can use the solution from the previous iteration as a warm start. We propose to impose a short subproblem time limit to prevent the solver from wasting time in proving optimality. In Section 4.6, we test two different variants of the selection rule. In one variant, two employees who could prevent the refusal of sub-request r^* are selected instead of just one. In another variant, two employees each with a refused sub-request associated with acceptance level a^* are selected and for each of those employees another employee is selected who can eliminate the corresponding refusal by a swap of shifts.

	Week 1							Week 2						
Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Ann	A_1	A_2	C_3	A_4	A_5				C_9	A_{10}	C_{11}	A_{12}	A_{13}	A_{14}
Bob	B_1	L_2	B_3		B_5	B_6	B_7	L_8	B_9				B_{13}	B_{14}
Dan	E_1	E_2	A_3	E_4	E_5			A_8	A_9	L_{10}	B_{11}			
Eva			E_3			A_6	A_7	E_8			A_{11}	E_{12}		
Gil				L_4					E_9	E_{10}	E_{11}	L_{12}		

Figure 4.8: Temporary schedule in the course of the matheuristic

4.5.2 Illustrative example

We applied the proposed matheuristic to the illustrative example introduced in Section 4.2.2. Figure 4.8 shows a temporary schedule in the course of the matheuristic for an iteration with acceptance level $a^* = 60$. In this schedule, 4 sub-requests are refused at the corresponding acceptance level.

- Sub-request 163: Ann has 6 consecutive working days
- Sub-request 214: Bob has an isolated day off
- Sub-request 337: Bob has no weekend off
- Sub-request 348: Eva's workload exceeds the target of 40 hours

In the next iteration, we apply the selection rule to define the subproblem.

1. Employee i^* = Eva is selected.
2. i^* = Eva has only one refused sub-request: $r^* = 348$
3. Sub-request $r^* = 348$ is of type 10, that means case 2 is applicable. Employee Dan is selected because he has the same skills as i^* = Eva.
4. Employee Gil is randomly selected.

In Figure 4.8, the three selected employees are enclosed by a frame. The subproblem consists of these employees and is solved to optimality by the solver. On the Friday of the second week, Dan cannot be assigned to shift E_{12} because of request type 7 (11 hours

between two consecutive shifts). However, Dan can be assigned to shift L_{12}, and Gil is assigned to shift E_{12}, and the refusal is eliminated. The resulting schedule corresponds to the optimal schedule shown in Figure 4.6.

4.6 Computational analysis

In this section, we evaluate the performance of the proposed approaches. In Section 4.6.1, we present the test instances. In Section 4.6.2, we describe the design of the analysis. In Section 4.6.3, we report and analyze the numerical results.

4.6.1 Test instances

Problem instances for the SAP-LAL are different from existing benchmark instances from the literature. For example, instances from the literature do not contain mapping functions that assign acceptance levels to deviations from the target values. We therefore generate SAP-LAL instances on the basis of real-world data that we obtained from the service provider. This has the advantage that a comparison with the service provider's software is possible. Our test instances include a test set of 45 systematically constructed instances and a real-world instance. We first describe the test set. All 45 instances of the test set have a planning horizon of four weeks. The types of requests to be considered in each instance are those presented in Table 4.1. The instances were constructed such that they differ with respect to the following three complexity parameters:

- The number of available employees NE: We generated instances with 10, 30, 50, 70, and 90 employees. Instances with 10 employees are considered small-sized, instances with 30 and 50 employees are considered medium-sized, and instances with 70 and 90 employees are considered large-sized. For each employee, we randomly selected a target workload of 80, 120, or 160 hours.

- The workload ratio WR: Given the target workloads of employees, the workload ratio determines the number of eight-hour shifts to be performed. The number of shifts is obtained by multiplying the sum of target workloads of all employees by WR and dividing the result by 8 (the length of a shift). We generated instances with a workload ratio of 0.9, 1, and 1.1. For each shift, we randomly determined the start time and the set of employees who have the required skills to perform it. The start times of shifts are equally distributed across the planning horizon.

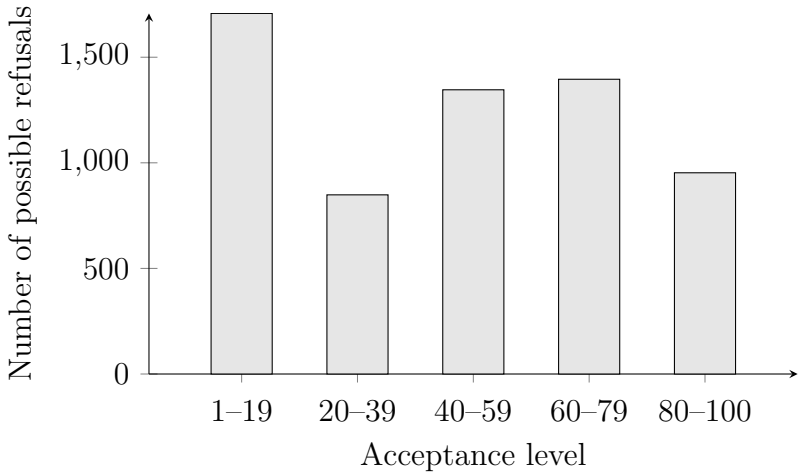


Figure 4.9: Average number of possible refusals per acceptance level over all instances

- Number of different acceptance levels NL. We generated instances with 10, 20 and 30 different acceptance levels. The number of acceptance levels corresponds to the number of optimizations to be performed in lexicographical order. First, a set containing NL different acceptance levels is generated. Thereby, the acceptance levels are equally distributed between 1 and 99. For each employee, the number of kinks in the mapping functions of request type 9 is 3 and of request types 10–13 is 10. All other mapping functions have a single kink only.

In total, we generated 45 instances, one instance for each possible combination of the three complexity parameters. Figure 4.9 visualizes the average number of possible refusals per acceptance level over all instances. The number of possible refusals varies from 848 to 1,707 between different ranges of acceptance levels.

The real-world instance stems from a client of our industry partner. It comprises 15 employees, a planning horizon of 28 days, the 13 request types presented in Table 4.1, and 23 variants of the request types presented in Table 4.1. These variants are structurally identical to the baseline request types. For example, the variants "at most 4 consecutive early shifts" and "at most 3 consecutive late shifts" are structurally identical to the requests of type 5 ("at most 5 consecutive work days").

4.6.2 Test design

We tested the following approaches:

- **BF:** Basic formulation.

- **AF:** Aggregated formulation.
- **$\mathbf{MH}_k$:** Baseline matheuristic as described in Section 4.5 with a subproblem time limit of 3 seconds. The subscript k indicates the size of the subproblems. We ran MH_k for $k = 4, 5, 6, 7, 8, 9$.
- **$\mathbf{MHR}_k$:** Matheuristic MH_k with a random employee selection rule. Under this rule, the employees are selected randomly with equal probability. We ran MHR_k for $k = 4, 5, 6, 7, 8, 9$.
- **$\mathbf{MHF}_k$:** Matheuristic MH_k with a fix-and-optimize strategy. Under this strategy, the subproblems are constructed by fixing decision variables in the complete model without removing them. We ran MHF_k for $k = 8$.
- **$\mathbf{MH60}_k$:** Matheuristic MH_k with a subproblem time limit of 60 seconds instead of 3 seconds. We ran $\mathrm{MH60}_k$ for $k = 8$.
- **$\mathbf{MH}_k^{1-2}$:** Matheuristic MH_k with a variant of the employee selection rule. Under this rule, one employee who has sub-request r^* refused is selected, and two more employees who could prevent the refusal of sub-request r^* are selected instead of just one. $k-3$ employees are selected randomly. We ran MH_k^{1-2} for $k = 8$.
- **$\mathbf{MH}_k^{2-2}$:** Matheuristic MH_k with a variant of the employee selection rule. Under this rule, two employees with a refused sub-request associated with acceptance level a^* are selected and for each of those employees another employee is selected who can eliminate the refusal by a swap of shifts. $k-4$ employees are selected randomly. We ran MH_k^{2-2} for $k = 8$.
- **SP:** Software package of our industry partner who reported the SAP-LAL.

All approaches except SP are implemented in AMPL and use the Gurobi Optimizer 6.5.1 as solver. For the exact approaches, we prescribed an acceptance level time limit of 300 seconds for the optimization of each individual acceptance level. For all variants of the matheuristics we used the three stopping criteria presented in Section 4.5. Thereby, the upper bound on the number of subproblems solved without improvement (IMP criterion) was set to 100 and the acceptance level time limit (CPU criterion) was set to 180 seconds. The computations were performed on a standard workstation with two 6-core Intel(R) Xeon(R) X5650 2.66GHz CPUs and 24GB RAM.

The quality of a solution to the SAP-LAL cannot be expressed by a single objective function value due to the lexicographic order of acceptance levels. Instead, we need to

compare the number of refused sub-requests for each acceptance level separately. As the exact approaches provide for each acceptance level a lower bound on the number of refusals, we can use the following three performance criteria to evaluate the exact approaches:

- **OPT:** Number of instances solved to optimality. The optimality of a solution is proven if and only if for each acceptance level, the lower bound on the number of refusals coincides with the number of refusals in the solution obtained.
- **PSO:** Mean average percentage of BLPs solved to optimality.
- **ALF:** Average of acceptance level of the first BLP that is not solved to optimality.

The above performance criteria cannot be used for evaluating the different variants of the matheuristic since they do not provide a lower bound on the number of refusals. It is possible, however, to rank different solutions by comparing the number of refusals for each acceptance level. We therefore use the following performance criteria to compare the matheuristic variants in terms of solution quality:

- **OPT*:** Number of instances for which the schedule obtained is optimal. For this criterion it is not necessary that the optimality has been proven. Note that we can only evaluate this criterion for the instances for which the optimal solution is known.
- **NBE:** Number of instances for which an approach found the best solution.
- **ARE:** Average number of refused sub-requests per employee.
- **AMR:** Average maximum number of refused sub-requests per employee.
- **AVR:** Average variance of the number of refused sub-requests per employee. This metric captures the fairness of a schedule. Schedules that are perceived as fair tend to have low AVR.

Note that criterion ARE does not take into account the lexicographic order between different acceptance levels. Nevertheless, it is used in practice to compare different solutions. In addition, all approaches are compared in terms of average CPU time requirement per instance in seconds (**CPU**).

4.6.3 Numerical results

In Subsection 4.6.3.1, we compare the results of the two exact approaches. In Subsection 4.6.3.2, we compare these results with the results of the baseline matheuristic. In Subsections 4.6.3.3–4.6.3.5, we analyze the results of different variants of the matheuristic and investigate the effectiveness of individual components of the matheuristic. In Subsection 4.6.3.6, we report the results obtained for the real-world instance.

4.6.3.1 Exact approaches

Table 4.5 reports the performance criteria for the two exact approaches BF and AF. The performance criteria are computed separately for small-, medium-, and large-sized instances and also for the entire test set. Both approaches are able to solve small- and medium-sized instances to optimality. For large-sized instances, the exact approaches were not able to prove optimality. However, the solution quality is still surprisingly high as around 80% of the binary-linear programs (BLPs) for large-sized instances are solved to optimality (see PSO values in Table 4.5), and that these BLPs correspond to the lowest acceptance levels. The 20% of the BLPs that are not solved to optimality correspond to high acceptance levels (above 78, see ALF values in Table 4.5). Together these results reflect high solution quality. The optimal solutions for BLPs associated with low acceptance levels are usually found within few seconds as shown in Figure 4.10. The CPU times are considerably higher for BLPs associated with higher acceptance levels than for BLPs associated with lower acceptance levels. The reason is, that in each BLP associated with acceptance level a^*, all sub-requests associated with an acceptance level $a \leq a^*$ need to be considered. Consequently, the number of sub-requests and thus the complexity increases with increasing value of the acceptance level.

It can be seen in Figure 4.10 that the CPU time requirement of the BLP with the lowest acceptance level is slightly higher than for the subsequently solved BLPs. This is because in the first BLP a feasible solution needs to be constructed from scratch which is not necessary in all other BLPs since the solution of the previous BLP can be used as a warm start. Formulation (AF) requires on average slightly more CPU time for the first BLP but slightly less CPU time for the other BLPs than formulation (BF).

Overall, both approaches (AF and BF) perform very similarly as can be seen from the last column of Table 4.5. The small performance difference can be explained by the fact that the aggregation techniques can only be applied to a small subset of constraints, namely those that refer to mapping functions with multiple kinks. For large-sized instances which contain more of those constraints, approach AF appears to be slightly

Table 4.5: Numerical results for exact approaches

		NE					
		10	30	50	70	90	All
		small	medium		large		
OPT	BF	8	2	0	0	0	10
	AF	7	2	1	0	0	10
CPU	BF	269	1,075	1,367	1,671	2,025	1,281
	AF	244	1,095	1,345	1,495	2,073	1,250
PSO	BF	98.5	86.7	80.9	78.1	78.9	84.6
	AF	99.1	85.7	82.6	83.0	81.1	86.3
ALF	BF	98.6	86.8	81.0	77.7	78.0	84.4
	AF	97.2	84.7	81.3	78.8	78.0	84.0

better than approach BF.

Next we study the impact of the complexity parameters WR and NL on the performance of the two exact approaches. Tables 4.6 and 4.7 state the performance criteria for groups of instances that have the same workload ratio (WR) and groups of instances that have the same number of acceptance levels (NL), respectively. It turns out that both complexity parameters affect the performance of both approaches in the same way. The higher the value of WR, the more shifts in relation to employees must be assigned which makes it more difficult to comply with sub-requests. This is reflected in Table 4.6 by the lower PSO and ALF values for $WR = 1.1$ as compared to $WR = 0.9$. Also, instances with $WR = 1.1$ require considerably more CPU time than instances with $WR = 0.9$.

Higher values of parameter NL do not affect the solution quality. The performance criteria OPT, PSO, and ALF have similar values for instances with different NL values. This is interesting because for instances with a high NL value, fewer refusals per acceptance level are possible as compared to instances with a low NL value. Apparently, even though the number of possible refusals is low, the difficulty of the instances remains the same. However, parameter NL affects the CPU time requirement. Instances with higher values of NL, i.e. with a larger number of different acceptance levels, require more CPU time because for each acceptance level, a separate BLP needs to be solved.

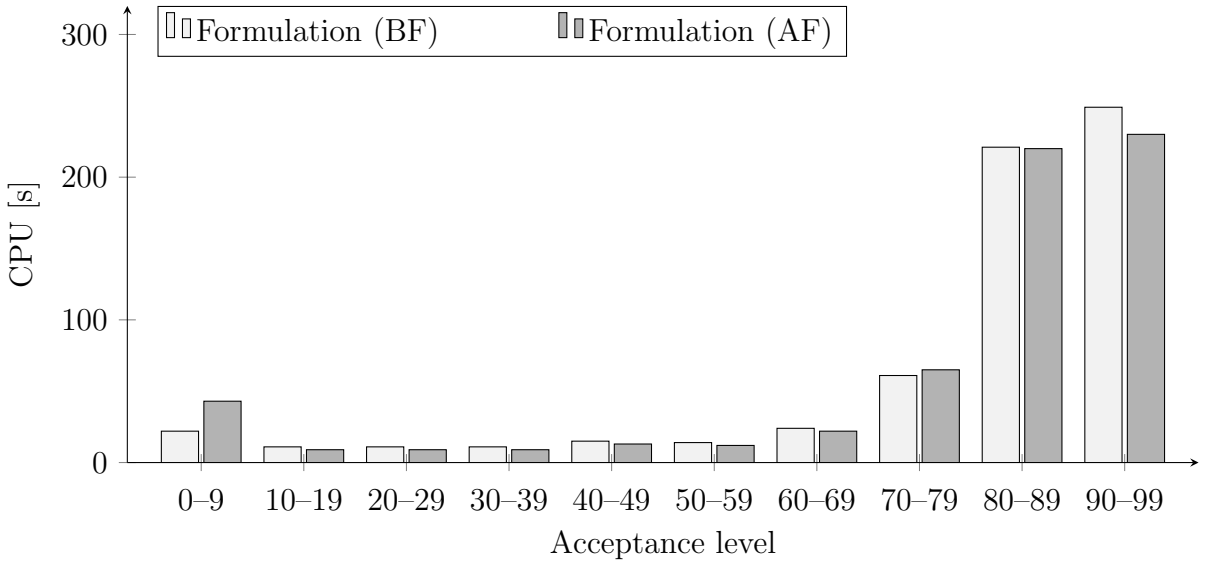


Figure 4.10: CPU time per acceptance level

Table 4.6: Impact of complexity parameter WR

		WR			
		0.9	1	1.1	All
OPT	BF	4	4	2	10
	AF	6	3	1	10
CPU	BF	1,023	1,206	1,615	1,281
	AF	889	1,212	1,650	1,250
PSO	BF	89.0	85.6	79.3	84.6
	AF	93.3	85.9	79.7	86.3
ALF	BF	88.1	85.6	79.5	84.4
	AF	89.3	84.0	78.7	84.0

Table 4.7: Impact of complexity parameter NL

		NL			
		10	20	30	All
OPT	BF	4	4	2	10
	AF	5	3	2	10
CPU	BF	634	1,266	1,945	1,281
	AF	642	1,313	1,796	1,250
PSO	BF	84.7	85.7	83.6	84.6
	AF	86.7	86.0	86.2	86.3
ALF	BF	84.7	85.0	83.5	84.4
	AF	86.0	83.0	83.0	84.0

4.6.3.2 Matheuristic: comparison with exact approaches

Table 4.8 lists for six performance criteria the results of the two exact approaches BF and AF and the results of the matheuristics MH_k with $k = 4, 5, 6, 7, 8, 9$. In this section, we focus on the comparison between the performance of the matheuristic and the performance of the two exact approaches. The following insights can be obtained from this comparison:

- All variants of the matheuristic are able to devise optimal solutions. Among the 12 instances for which optimal solutions are known, MH_8 provides an optimal solution for 11 instances.
- The matheuristic variants considerably outperform the exact approaches for medium- and large-sized instances. This is reflected best by performance criterion ARE. While the ARE values of both exact approaches increase considerably for medium- and large-sized instances, they remain at a low level for all variants of the matheuristic. This demonstrates that all matheuristic variants find high-quality solutions for large instances.
- With respect to performance criteria AMR and AVR, all variants of the matheuristic clearly outperform the exact approaches, i.e., they tend to generate schedules with higher fairness. Figure 4.11 shows for the three approaches AF, BF, and MH_8, boxplots that represent the distribution of the number of refusals among employees for a specific instance with 90 employees. The thick horizontal line marks the

median of the distribution, the bottom and top of the box correspond to the first and third quartiles, and the whiskers represent the minimum and maximum number of refusals. A possible explanation for this outperformance of the matheuristic is the fact that the employees with a large number of refusals are more likely to be selected by the employee selection rule than employees with a low number of refusals. Since only selected employees have their refusals reverted, the guided selection leads to a more balanced distribution of the number of refusals.

- All variants of the matheuristic require less CPU time than both exact approaches.
- Interestingly, although in the direct comparison formulations AF and BF performed equally well, formulation AF outperforms formulation BF with respect to all performance criteria shown in Table 4.8.

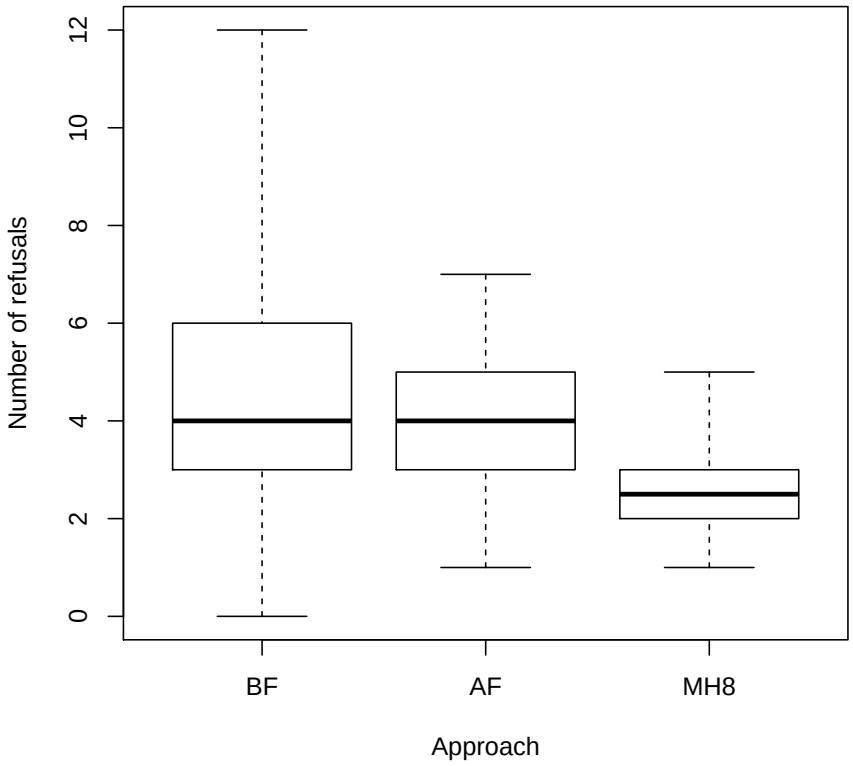


Figure 4.11: Distributions of number of refusals among employees for approach BF, AF, and MH_8 for an instance with 90 employees

4.6.3.3 Matheuristic: impact of the size of the subproblems

The goal of this section is to study the impact of the size of the subproblems on the performance of the matheuristic. The size of the subproblems is determined by parameter

Table 4.8: Comparison of variants of matheuristic

		NE					
		10	30	50	70	90	All
		small	medium		large		
OPT*	BF	8	2	0	0	0	10
	AF	8	2	1	0	0	11
	MH_4	3	0	0	0	0	3
	MH_5	3	0	0	0	0	3
	MH_6	5	1	0	0	0	6
	MH_7	6	0	1	0	0	7
	MH_8	7	3	1	0	0	11
	MH_9	5	3	0	0	0	8
NBE	BF	8	3	0	1	1	13
	AF	9	2	1	2	0	14
	MH_4	3	1	0	0	0	4
	MH_5	3	0	1	0	0	4
	MH_6	5	2	2	0	0	9
	MH_7	6	2	5	3	0	16
	MH_8	7	5	3	1	4	20
	MH_9	5	4	0	2	4	15
ARE	BF	2.66	2.94	3.76	4.30	9.38	4.61
	AF	2.63	3.00	3.49	4.34	8.45	4.38
	MH_4	2.52	2.46	2.46	2.73	2.97	2.63
	MH_5	2.50	2.34	2.38	2.54	2.68	2.49
	MH_6	2.62	2.27	2.29	2.48	2.63	2.46
	MH_7	2.60	2.25	2.32	2.43	2.57	2.43
	MH_8	2.62	2.29	2.32	2.45	2.54	2.45
	MH_9	2.77	2.40	2.43	2.59	2.64	2.57
AMR	BF	5.00	5.56	7.44	8.56	15.22	8.36
	AF	4.89	5.89	7.56	9.11	13.33	8.16
	MH_4	4.67	4.78	5.22	6.22	6.78	5.53
	MH_5	4.33	4.22	5.00	5.22	6.00	4.96
	MH_6	4.89	4.22	4.56	5.67	6.11	5.09
	MH_7	5.00	4.33	4.67	5.33	5.78	5.02
	MH_8	5.33	4.00	4.89	5.11	6.00	5.07
	MH_9	4.67	4.67	4.89	5.67	6.00	5.18
AVR	BF	2.41	1.75	3.50	3.53	5.17	3.27
	AF	2.25	2.09	3.13	3.68	4.41	3.11
	MH_4	2.17	1.48	1.41	1.83	1.96	1.77
	MH_5	1.97	0.97	1.24	1.41	1.73	1.47
	MH_6	2.16	1.10	1.10	1.39	1.68	1.49
	MH_7	2.83	1.01	1.23	1.30	1.48	1.57
	MH_8	2.69	0.82	1.30	1.43	1.58	1.56
	MH_9	2.26	1.25	1.32	1.50	1.63	1.59
CPU	BF	269	1,075	1,367	1,671	2,025	1,281
	AF	244	1,095	1,345	1,495	2,073	1,250
	MH_4	66	217	686	1,363	2,049	876
	MH_5	128	305	710	1,267	1,932	868
	MH_6	207	418	699	1,199	1,837	872
	MH_7	345	471	742	1,131	1,753	889
	MH_8	452	506	757	1,160	1,692	913
	MH_9	479	541	790	1,123	1,617	910

k. The impact of k is analyzed based on the results given in Table 4.8 from which we draw the following conclusions:

- Among the different matheuristic variants, variant MH_8 delivers the best overall results in terms of solution quality. This variant achieved the best NBE and OPT^* value as can be seen from the last column in the table.
- For large-sized instances, variants with $k \geq 7$ deliver better results than variants with $k \leq 6$. This shows that in order to reduce the number of refusals in large-sized instances shift swaps are required that involve multiple employees.
- With respect to performance criteria AMR and AVR, no significant differences can be observed between the variants of the matheuristic.
- The CPU time requirement of the matheuristic depends on k and the size of the problem instances. For small- and medium-sized instances, usually the stopping criterion IMP (no improvement was achieved for 100 consecutive subproblems) is met first. As variants with a low value of k generally require less time per subproblem, they are faster for small- and medium-sized instances than variants with a large value of k. For large-sized instances, usually the stopping criterion CPU (the acceptance level time limit has been reached) is met first. As variants with a large value of k are often able to revert all refusals associated with a specific acceptance level, they can continue with the next acceptance level while variants with a small value of k are often not able to revert all refusals and thus need to wait for criterion CPU to be met. Under this setting, variants with a large value of k can be faster for large-sized instances than variants with a small value of k.

We also investigated the influence of the strategy to formulate the subproblems without redundant constraints and variables on the performance of the matheuristic. Due to this strategy, the size of the subproblems is reduced considerably. In Table 4.9, we compare the results of MH_8 with the results of version MHF_8 which uses a fix-and-optimize strategy, i.e, the subproblems are constructed by fixing decision variables in the complete model without removing them. Here we report the results only for $k = 8$ as we obtained similar results for other values of k. Approach MH_8 overall outperforms MHF_8 both in terms of solution quality and CPU time requirement. The outperformance is most distinct for medium- and large-sized instances.

Table 4.9: Impact of strategy to formulate the subproblems without redundant constraints and variables

		NE					
		10	30	50	70	90	All
		small	medium		large		
OPT*	MH_8	7	3	1	0	0	11
	MHF_8	6	2	0	0	0	8
NBE	MH_8	8	6	8	9	9	40
	MHF_8	7	5	2	0	0	14
ARE	MH_8	2.62	2.29	2.32	2.45	2.54	2.45
	MHF_8	2.66	2.27	2.34	2.62	3.58	2.69
AVR	MH_8	2.69	0.82	1.30	1.43	1.58	1.56
	MHF_8	2.70	1.09	1.25	1.56	3.01	1.92
CPU	MH_8	452	506	757	1,160	1,692	913
	MHF_8	437	595	1,047	1,782	3,260	1,424

4.6.3.4 Matheuristic: impact of the employee selection rule

In this section, we examine the impact of the employee selection rule. In Table 4.10, we compare the results of MH_k with $k = 4, 5, 6, 7, 8, 9$ to the results of a simplified version MHR_k which does not use the employee selection rule and instead selects employees randomly. The last two columns of the table contain for both variants the average number of subproblems that were passed to the solver (NSP). Approach MH_k clearly outperforms MHR_k for all values of k both in terms of solution quality and CPU time requirement. The employee selection rule is most effective for small values of k. This is probably due to the fact that for small values of k, less employees are randomly selected to be included in the subproblem. If k is small, only few combinations of employees can eliminate refusals. These combinations are rarely found by a random selection. Approach MH_k requires less time because the employee selection rule effectively identifies subproblems that lead to a reduction in the number of refusals. A random selection of employees often results in subproblems that do not lead to a reduction in the number of refusals. Consequently, approach MH_k performs fewer iterations (see the NSP values in Table 4.10).

We also investigated other specific employee selection rules. In Table 4.11, we compare

Table 4.10: Impact of the employee selection rule

k	NBE		CPU		NSP	
	MH_k	MHR_k	MH_k	MHR_k	MH_k	MHR_k
4	43	5	876	1,403	1,172	1,820
5	35	13	868	1,339	967	1,425
6	33	18	872	1,267	795	1,163
7	32	20	889	1,253	700	977
8	34	20	913	1,228	609	853
9	31	26	910	1,187	573	776

the results to the basic variant of the matheuristic MH_8. In MH_8^{2-2}, two employees with a refusal are selected, and for each employee at least one other employee which can prevent the refusal. In MH_8^{1-2}, only one employee with a refusal is selected, but at least two employees which can prevent the refusal of the first one. Both, MH_8^{2-2} and MH_8^{1-2}, overall outperform the basic variant MH_8 in terms of solution quality which emphasizes the effectiveness of the employee selection rule.

4.6.3.5 Matheuristic: impact of the subproblem time limit

In this section, we analyze the impact of the subproblem time limit on the performance of the matheuristic. In Table 4.12, we compare the results of MH_8 (with a default subproblem time limit of 3 seconds) with the results of approach $MH60_8$ which uses a subproblem time limit of 60 seconds. Interestingly, increasing the subproblem time limit to 60 seconds does not improve the solution quality. In contrast, the solution quality decreases with the increased subproblem time limit. This is because for most subproblems the solver finds the best solution in few seconds but does not terminate until the optimality of this solution is proven. By increasing the subproblem time limit, fewer subproblems are solved for each acceptance level because of the acceptance level time limit.

4.6.3.6 Numerical results for real-world instance

We applied the best exact approach (AF) and the best variant of the matheuristic (MH_8) to a real-world instance and compared the results to those of the problem-specific software package of our industry partner (SP). Table 4.13 lists for all three approaches the results for each acceptance level. The values in bold indicate for each approach up to which acceptance level the number of refusals is identical to the number of refusals in the best

Table 4.11: Comparison of different variants of the employee selection rule

		NE					
		10	30	50	70	90	All
		small	medium		large		
OPT*	MH_8	7	3	1	0	0	11
	MH_8^{2-2}	6	2	1	0	0	9
	MH_8^{1-2}	6	3	0	0	0	9
NBE	MH_8	7	5	2	3	2	19
	MH_8^{2-2}	6	2	7	2	4	21
	MH_8^{1-2}	8	7	1	4	3	23
ARE	MH_8	2.62	2.29	2.32	2.45	2.54	2.45
	MH_8^{2-2}	2.66	2.28	2.31	2.45	2.50	2.44
	MH_8^{1-2}	2.58	2.30	2.34	2.44	2.57	2.44
AVR	MH_8	2.69	0.82	1.30	1.43	1.58	1.56
	MH_8^{2-2}	2.54	0.95	1.14	1.25	1.46	1.47
	MH_8^{1-2}	2.38	0.97	1.29	1.34	1.61	1.52
CPU	MH_8	452	506	757	1,160	1,692	913
	MH_8^{2-2}	458	493	723	1,049	1,518	848
	MH_8^{1-2}	444	521	780	1,159	1,698	920
NSP	MH_8	454	558	710	703	620	609
	MH_8^{2-2}	452	562	673	634	541	572
	MH_8^{1-2}	460	582	741	694	624	620

Table 4.12: Impact of subproblem time limit

		NE					
		10	30	50	70	90	All
		small	medium		large		
OPT	MH_8	7	3	1	0	0	11
	$MH60_8$	6	2	0	0	0	8
NBE	MH_8	7	8	9	8	7	39
	$MH60_8$	8	3	1	1	2	15
ARE	MH_8	2.62	2.29	2.32	2.45	2.54	2.45
	$MH60_8$	2.54	2.77	3.21	4.45	5.25	3.65
AVR	MH_8	2.69	0.82	1.30	1.43	1.58	1.56
	$MH60_8$	2.71	1.55	1.97	4.30	5.16	3.14
CPU	MH_8	452	506	757	1160	1692	913
	$MH60_8$	496	585	815	1184	1690	954
NSP	MH_8	454	558	710	703	620	609
	$MH60_8$	368	482	596	658	600	541

solution found by approach MH_8. For approach AF, the table reports the number of refused sub-requests (NR), the lower bound on the number of refused sub-requests (LB) and the CPU time requirement (CPU) in seconds. For approach MH_8, the table reports the number of refused sub-requests (NR), the number of subproblems passed to the solver (NSP), and the CPU time requirement (CPU). For the software package of our industry partner, we report the number of refused sub-requests per acceptance level. The CPU time requirement of approach SP cannot be compared to the CPU time requirement of the other approaches as approach SP was run by the industry partner on a different computer. For acceptance levels below 46, all approaches have the same number of refusals per acceptance level. The solutions obtained by approach AF and MH_8 have only three refusals at acceptance level 46 whereas approach SP has four refusals. As one refusal associated with a lower acceptance level is worse than any number of refusals with higher acceptance level, the solutions obtained by approaches AF and MH_8 are better than the solution obtained by approach SP. The best solution is obtained by approach MH_8, because it has 14 refusals associated with acceptance level 51 compared to 15 refusals in

the solution obtained by approach AF.

The comparison was quite important for the service provider. For the first time, they were able to benchmark their own approach and get an understanding of the quality of their solutions. Moreover, they were able to study characteristics of optimal solutions for small- and medium-sized instances. The (optimal) schedules generated by the proposed approaches were analyzed systematically by the service provider to find opportunities for improving their approach. New versions of the software have been released based on the results of this analysis.

4.7 Conclusions

We introduced a real-world staff assignment problem that was reported to us by a Swiss provider of employee scheduling software. This provider has developed a framework that helps decision makers to specify trade-offs between different requests such as employees' personal preferences by means of hierarchically-ordered acceptance levels. The framework gives rise to a new type of staff assignment problem for which existing solution techniques are not appropriate. We proposed a novel lexicographic goal programming approach for solving small instances to optimality, and we developed a matheuristic for large-scale instances. The matheuristic iteratively improves an initial solution by solving subproblems which involve only subsets of employees. The subsets are defined according to a new and effective employee selection rule. The performance of the exact and heuristic approaches is evaluated based on a collection of problem instances that we derived from real-world data.

The software provider involved in this research benefits from our research in two ways. First, the solutions generated by our approach enable the provider to evaluate the current performance of its software. Second, the provider gains insights into the structure of optimal solutions which is helpful for improving the performance of its software.

In future research, we plan to develop further variants of the matheuristic. A promising idea is to vary the size of the subproblems dynamically, i.e., increase the size after a predefined number of iterations without improvements. Furthermore, according to the software provider, most of their clients consider a fair distribution of refusals among employees to be a desirable objective. In Rihm and Baumann (2015a), we present model extensions that allow to improve an existing schedule in terms of fairness without deteriorating its quality with regard to refused requests. Balancing both fairness and number of refusals is still to be addressed.

Table 4.13: Numerical results for real-world instance

AL	AF			MH_8			SP
	NR	LB	CPU	NR	CPU	NSP	NR
1	**2**	2	1	**2**	18	106	**2**
5	**5**	5	1	**5**	17	106	**5**
20	**0**	0	2	**0**	3	13	**0**
21	**0**	0	0	**0**	1	5	**0**
25	**0**	0	1	**0**	1	4	**0**
26	**0**	0	1	**0**	0	1	**0**
32	**0**	0	4	**0**	7	6	**0**
34	**0**	0	7	**0**	32	24	**0**
37	**0**	0	70	**0**	4	3	**0**
38	**4**	4	77	**4**	40	111	**4**
40	**0**	0	90	**0**	1	4	**0**
41	**0**	0	0	**0**	0	1	**0**
43	**0**	0	3	**0**	1	2	**0**
44	**0**	0	2	**0**	1	2	**0**
45	**0**	0	29	**0**	5	2	**0**
46	**3**	3	55	**3**	122	123	4
47	**0**	0	197	**0**	15	7	0
49	**0**	0	2	**0**	0	1	0
50	**3**	3	222	**3**	66	138	4
51	15	10	300	**14**	180	114	14
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
97	0	0	5	**0**	1	3	0
99	1	0	300	**5**	131	103	4
Total	60	33	4,482	71	2,061	2,357	68

Bibliography

Aickelin, U., Dowsland, K., 2000. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Journal of Scheduling 3 (3), 139–153.

Aickelin, U., Dowsland, K. A., 2004. An indirect genetic algorithm for a nurse-scheduling problem. Computers & Operations Research 31 (5), 761–778.

Al-Yakoob, S., Sherali, H., 2007. Mixed-integer programming models for an employee scheduling problem with multiple shifts and work locations. Annals of Operations Research 155 (1), 119–142.

Azaiez, M. N., Al Sharif, S., 2005. A 0-1 goal programming model for nurse scheduling. Computers & Operations Research 32 (3), 491–507.

Bai, R., Burke, E. K., Kendall, G., Li, J., McCollum, B., 2010. A hybrid evolutionary approach to the nurse rostering problem. IEEE Transactions on Evolutionary Computation 14 (4), 580–590.

Ball, M. O., 2011. Heuristics based on mathematical programming. Surveys in Operations Research and Management Science 16 (1), 21–38.

Bard, J. F., Wan, L., 2006. The task assignment problem for unrestricted movement between workstation groups. Journal of Scheduling 9 (4), 315–341.

Beaulieu, H., Ferland, J. A., Gendron, B., Michelon, P., 2000. A mathematical programming approach for scheduling physicians in the emergency room. Health Care Management Science 3 (3), 193–200.

Berrada, I., Ferland, J. A., Michelon, P., 1996. A multi-objective approach to nurse scheduling with both hard and soft constraints. Socio-Economic Planning Sciences 30 (3), 183–193.

Bertels, S., Fahle, T., 2006. A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. Computers & Operations Research 33 (10), 2866–2890.

Bester, M., Nieuwoudt, I., Van Vuuren, J. H., 2007. Finding good nurse duty schedules: a case study. Journal of Scheduling 10 (6), 387–405.

Bixby, R. E., 2012. A brief history of linear and mixed-integer programming computation. Documenta Mathematica, 107–121.

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.), Hybrid Metaheuristics: 6th International Workshop on Hybrid Metaheuristics, Udine. Springer, Berlin, Heidelberg, pp. 171–177.

Chang, C.-T., 2006. Mixed binary interval goal programming. Journal of the Operational Research Society 57, 469–473.

Chang, C.-T., Lin, T.-C., 2009. Interval goal programming for s-shaped penalty function. European Journal of Operational Research 199, 9–20.

Charnes, A., Collomb, B., 1972. Optimal economic stabilization policy: Linear goal-interval programming models. Socio-Economic Planning Sciences 6 (4), 431–435.

Charnes, A., Cooper, W. W., Ferguson, R. O., 1955. Optimal estimation of executive compensation by linear programming. Management Science 1, 138–151.

Charnes, A., Cooper, W. W., Harrald, J., Karwan, K. R., Wallace, W. A., 1976. A goal interval programming model for resource allocation in a marine environmental protection program. Journal of Environmental Economics and Management 3, 347–362.

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, 512–522.

Cordeau, J.-F., Laporte, G., Pasin, F., Ropke, S., 2010. Scheduling technicians and tasks in a telecommunications company. Journal of Scheduling 13 (4), 393–409.

De Bruecker, P., Van den Bergh, J., Beliën, J., Demeulemeester, E., 2015. Workforce planning incorporating skills: State of the art. European Journal of Operational Research 243 (1), 1–16.

Della Croce, F., Salassa, F., 2014. A variable neighborhood search based matheuristic for nurse rostering problems. Annals of Operations Research 218 (1), 185–199.

Dowsland, K. A., 1998. Nurse scheduling with tabu search and strategic oscillation. European Journal of Operational Research 106 (2), 393–407.

Eiselt, H. A., Marianov, V., 2008. Employee positioning and workload allocation. Computers & Operations Research 35 (2), 513–524.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D., 2004. An annotated bibliography of personnel scheduling and rostering. Annals of Operations Research 127 (1–4), 21–144.

Falasca, M., Zobel, C., Ragsdale, C., 2011. Helping a small development organization manage volunteers more efficiently. Interfaces 41 (3), 254–262.

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming 98 (1–3), 23–47.

Ignizio, J., 2004. Optimal maintenance headcount allocation: an application of chebyshev goal programming. International Journal of Production Research 42 (1), 201–210.

Jones, D., Tamiz, M., 1995. Expanding the flexibility of goal programming via preference modelling techniques. Omega 23 (1), 41–48.

Jones, D., Tamiz, M., 2010. Goal programming variants. In: Practical Goal Programming. Springer, pp. 11–22.

Jones, D., Tamiz, M., 2016. A review of goal programming. In: Greco, S., Ehrgott, M., Figueira, J. R. (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys. Springer, New York, pp. 903–926.

Jones, D. F., Mirrazavi, S. K., Tamiz, M., 2002. Multi-objective meta-heuristics: An overview of the current state-of-the-art. European Journal of Operational Research 137 (1), 1–9.

Jones, D. F., Tamiz, M., 2002. Goal programming in the period 1990–2000. In: Ehrgott, M., Gandibleux, X. (Eds.), Multiple criteria optimization — state of the art annotated bibliographic surveys. Kluwer Academic Publishers, Dordrecht.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., et al., 2011. MIPLIB 2010. Mathematical Programming Computation 3 (2), 103–163.

Kopanos, G. M., Méndez, C. A., Puigjaner, L., 2010. MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. European Journal of Operational Research 207 (2), 644–655.

Kvanli, A. H., 1980. Financial planning using goal programming. Omega 8, 207–218.

Lodi, A., 2010. Mixed integer programming computation. In: Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., Wolsey, L. A. (Eds.), 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art. Springer, Berlin, Heidelberg, pp. 619–645.

Louly, M. A. O., 2013. A goal programming model for staff scheduling at a telecommunications center. Journal of Mathematical Modelling and Algorithms in Operations Research 12 (2), 167–178.

Maenhout, B., Vanhoucke, M., 2008. Comparison and hybridization of crossover operators for the nurse scheduling problem. Annals of Operations Research 159 (1), 333–353.

Maniezzo, V., Stützle, T., Voss, S., 2009. Matheuristics: hybridizing metaheuristics and mathematical programming. Springer, New York, USA.

Mihaylov, M., Smet, P., Van Den Noortgate, W., Vanden Berghe, G., 2016. Facilitating the transition from manual to automated nurse rostering. Health Systems 5 (2), 120–131.

Parr, D., Thompson, J. M., 2007. Solving the multi-objective nurse scheduling problem with a weighted cost function. Annals of Operations Research 155 (1), 279–288.

Raidl, G. R., Puchinger, J., 2008. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: Blum, C., Aguilera, M. J. B., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics: An Emerging Approach to Optimization. Springer, Berlin, Heidelberg, pp. 31–62.

Rihm, T., Baumann, P., 2015a. Improving fairness in staff assignment: An approach for lexicographic goal programming. In: Magnanti, T., Chai, K., Jiao, R., Chen, S., Xie, M. (Eds.), Proceedings of the 2015 IEEE International Conference on Industrial Engineering and Engineering Management. Singapore, pp. 1247–1251.

Rihm, T., Baumann, P., 2015b. A lexicographic goal programming approach for staff assignment with acceptance levels. In: Hanzálek, Z., Kendall, G., McCollum, B., Šůcha,

P. (Eds.), Proceedings of the 7th Multidisciplinary International Conference on Scheduling: Theory and Applications. Prague, pp. 526–540.

Romero, C., 2004. A general structure of achievement function for a goal programming model. European Journal of Operational Research 153, 675–686.

Romero, C., 2014. Handbook of critical issues in goal programming. Pergamon Press, Oxford.

Smet, P., Bilgin, B., De Causmaecker, P., Vanden Berghe, G., 2014a. Modelling and evaluation issues in nurse rostering. Annals of Operations Research 218 (1), 303–326.

Smet, P., Vanden Berghe, G., 2012. A matheuristic approach to the shift minimisation personnel task scheduling problem. In: Kjenstad, D., Riise, A., Nordlander, T. E., McCollum, B., Burke, E. (Eds.), Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling. Son, pp. 145–160.

Smet, P., Wauters, T., Mihaylov, M., Vanden Berghe, G., 2014b. The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. Omega 46, 64–73.

Tamiz, M., Jones, D. F., El-Darzi, E., 1995. A review of goal programming and its applications. Annals of Operations Research 58, 39–53.

Topaloglu, S., 2006. A multi-objective programming model for scheduling emergency medicine residents. Computers & Industrial Engineering 51 (3), 375–388.

Valls, V., Pérez, Á., Quintanilla, S., 2009. Skilled workforce scheduling in service centres. European Journal of Operational Research 193 (3), 791–804.

Van den Bergh, J., Belïen, J., De Bruecker, P., Demeulemeester, E., 2013. Personell scheduling: a literature review. European Journal of Operational Research 226, 367–385.