J Sched
DOI 10.1007/s10951-017-0525-1

Staff assignment with lexicographically ordered acceptance levels

Tom Rihm[1] · **Philipp Baumann**[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.

✉ Philipp Baumann
philipp.baumann@pqm.unibe.ch

Tom Rihm
tom.rihm@pqm.unibe.ch

[1] Department of Business Administration, University of Bern, Schützenmattstrasse 14, 3012 Bern, Switzerland

Keywords Employee scheduling · Staff assignment · Real-world problem · Goal programming · Binary linear programming · Matheuristic

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 he or she 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 are 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,…,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 sub-requests 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 Sect. 2, we formally introduce the SAP-LAL and provide an illustrative example. In Sect. 3, we review the literature on employee scheduling. In Sects. 4 and 5, we describe the exact solution approach and the matheuristic, respectively. In Sect. 6, we report the design and the results of our computational analysis. Section 7 concludes the paper with a summary and directions for future research.

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 Sect. 2.1 and provide an illustrative example in Sect. 2.2.

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 to perform more than one shift, he or she 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 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 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, he or she generally prefers to have two consecutive days off (e.g., Thursday and Friday) rather than two isolated (nonconsecutive) days off (e.g., Tuesday and Friday). Requests of type 6 allow to express this preference.

Table 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 h 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

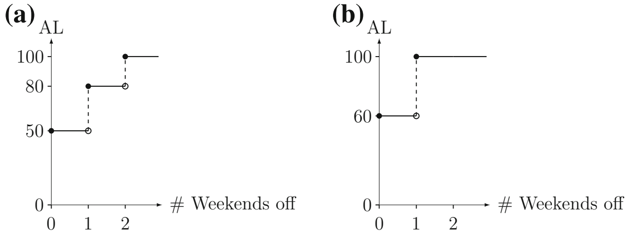


Fig. 1 Mapping functions for requests of type 9

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,…,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 1 shows two possible mapping functions for a request of type 9. Figure 1a 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 1b 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 Fig. 1a than for an employee with a mapping function as the one shown in Fig. 1b. 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.

2.2 Illustrative example

The planning horizon of the illustrative example spans 2 weeks. There are five different types of shifts: A, B, C, E (early shift), and L (late shift). Figure 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 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 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 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 2 also contains the acceptance lev-

Fig. 2 Illustrative example: shifts that need to be performed

Shift		Compatible employees	Week 1							Week 2						
Start	End		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}			

Table 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	Figure 1a
	337–339	Bob, Eva, Gil	Figure 1b
10	340–342	Ann, Bob, Dan	Figure 3a
	343–344	Eva, Gil	Figure 3b
11	345–347	Ann, Bob, Dan	Figure 3a
	348–349	Eva, Gil	Figure 3b
12	350–352	Dan, Eva, Gil	Figure 4a
13	353–355	Bob, Dan, Gil	Figure 4b

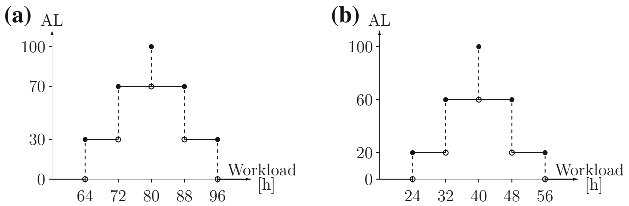


Fig. 3 Mapping functions of request types 10 and 11

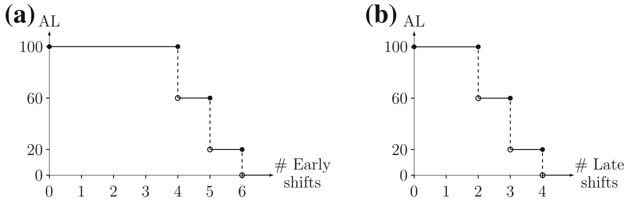


Fig. 4 Mapping functions of request types 12 and 13

els that are associated with refusing a request. Figures 1, 3, and 4 visualize the mapping functions of request types 9–13.

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 3.1–3.3 contain a description of popular techniques from each group and discuss the difficulties that arise when applying these techniques to the SAP-LAL.

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 generation, 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 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 nonlinear 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 underlying 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 h, 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 h. 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 h. Existing lexicographic goal programming approaches would always select schedule A or B, but never schedule C, although this appears to be the most favorable schedule. The existing weighted goal programming approaches and their extensions allow for a more accurate modeling 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 Sect. 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.

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 are 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 nonsmooth, 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.

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 behavior 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 Sect. 6, we demonstrate that selecting employees specifically is indeed superior to selecting employees randomly.

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 Sect. 4.1). In the second phase, a lexicographic goal program is formulated to iteratively optimize the sub-requests (see Sect. 4.2). In Sect. 4.3, we illustrate the exact solution approach by means of the illustrative example that we introduced in Sect. 2.2.

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

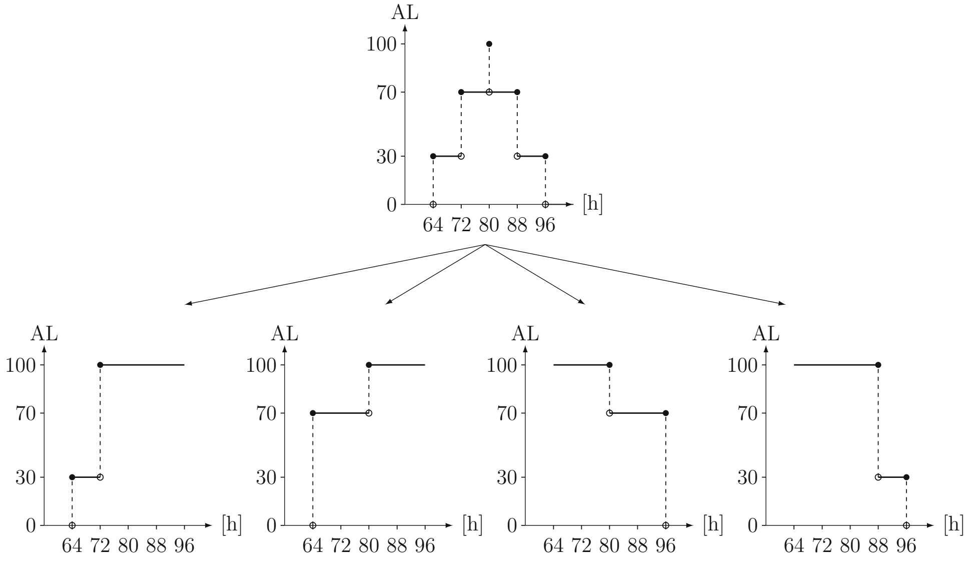


Fig. 5 Decomposition of a request into four sub-requests

level. It follows that each sub-request can be either 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 5 illustrates the decomposition of the workload request of Fig. 3a. This request can be refused to different degrees and is therefore decomposed into four sub-requests. The first sub-request (bottom left plot in Fig. 5) is refused when the workload is less than 72 h. Such a refusal is associated with an acceptance level of 30. The second sub-request (second plot from the left) is refused when the workload is less than 80 h. 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 h or of more than 96 h is infeasible. After the decomposition, each of the four sub-requests is associated with exactly one acceptance level, so a clear order of the sub-requests results.

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 Sect. 4.2.1, formulate the lexicographic goal program in Sect. 4.2.2, and present aggregation techniques for reducing the number of constraints in Sect. 4.2.3.

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

4.2.2 Model formulation

The model presented below covers the three types of critical requests and the ten types of noncritical requests presented in Table 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 (1) bound the number of refused sub-requests for all acceptance levels $a < a^*$ to ensure that the results from previous optimizations are preserved.

$$\sum_{r \in R_a} z_r \leq v_a \qquad (a \in A : a < a^*) \tag{1}$$

Constraints (2) address the requests of type 1. They ensure that each employee $i \in I$ is assigned to at most one shift each day $d \in D$.

$$\sum_{s \in S_{id}} x_{is} \leq 1 \qquad (i \in I,\ d \in D) \tag{2}$$

Constraints (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.

$$\sum_{i \in I_s} x_{is} = 1 \qquad (s \in S) \tag{3}$$

Constraint (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.

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

Constraint (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.

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

Constraint (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 $r \in R^6$.

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

Constraints (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_{id}^b 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 $r \in R^7$.

$$x_{i_r s_1} + x_{i_r s_2} \leq t_r + g_r z_r \quad (r \in R^7, (s_1, s_2) \in S_{i_r d_r}^b) \tag{7}$$

Constraints (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 $r \in R^8$.

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

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

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

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

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

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 (11) cover request type 11, which is intended to prevent that the actual workload of employees falls below the respective target workloads.

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

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

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

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

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

Finally, all variables are binary.

$$x_{is} \in \{0, 1\} \qquad (i \in I_s,\ s \in S) \tag{14}$$

$$y_{iw} \in \{0, 1\} \qquad (i \in I,\ w \in W) \tag{15}$$

$$z_r \in \{0, 1\} \qquad (r \in R) \tag{16}$$

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 (17) aggregate constraints (9).

$$\sum_{w \in W} y_{i_r w} \geq \max_{r \in R_i^9} (t_r) + \sum_{r \in R_i^9} h_r z_r \qquad (i \in I) \tag{17}$$

Analogously, we aggregate constraints (10), (11), (12) and (13):

$$\sum_{d \in D} \sum_{s \in S_{i_r d}} l_s x_{i_r s} \leq \min_{r \in R_i^{10}} (t_r) + \sum_{r \in R_i^{10}} h_r z_r \qquad (i \in I) \tag{18}$$

$$\sum_{d\in D}\sum_{s\in S_{i_r d}} l_s x_{i_r s} \geq \max_{r\in R_i^{11}} (t_r) + \sum_{r\in R_i^{11}} h_r z_r \qquad (i \in I) \qquad (19)$$

$$\sum_{d\in D}\sum_{s\in S^E_{i_r d}} x_{i_r s} \leq \min_{r\in R_i^{12}} (t_r) + \sum_{r\in R_i^{12}} h_r z_r \qquad (i \in I) \qquad (20)$$

$$\sum_{d\in D}\sum_{s\in S^L_{i_r d}} x_{i_r s} \leq \min_{r\in R_i^{13}} (t_r) + \sum_{r\in R_i^{13}} h_r z_r \qquad (i \in I) \qquad (21)$$

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.

$$\sum_{d\in D}\sum_{s\in S_{id}} l_s x_{is} = \min_{r\in R_i^{10}} (t_r) + \sum_{r\in R_i^{10}\cup R_i^{11}} h_r z_r \qquad (i \in I) \qquad (22)$$

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 Sect. 7, we compare the performance of the basic model formulation (BF) of Sect. 4.2.2 with the performance of the aggregated model formulation (AF) of Sect. 4.2.3:

$$(BF)\begin{cases}\text{Min} \displaystyle\sum_{r\in R_{a*}} z_r \\ \text{s.t. } (1)-(16)\end{cases} \qquad (AF)\begin{cases}\text{Min} \displaystyle\sum_{r\in R_{a*}} z_r \\ \text{s.t. } (1)-(8) \\ \quad (14)-(17) \\ \quad (20)-(22)\end{cases}$$

4.3 Illustrative example

In this section, we apply the exact approach to the illustrative example introduced in Sect. 2.2. Table 3 reports the result of the decomposition phase. The requests have been decomposed into 373 sub-requests. The fourth column of Table 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 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 6 depicts an optimal schedule for the illustrative example. All refused sub-requests are listed in Table 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 (≪1 s).

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 Sect. 5.1, we describe the matheuristic in detail. In Sect. 5.2, we apply the matheuristic to the illustrative example that we introduced in Sect. 2.2.

5.1 Matheuristic: description

The basic idea of the matheuristic is to iteratively improve an initial solution by reassigning groups of employees (see Fig. 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 behavior 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 non-

Table 3 Sub-requests for the illustrative example

Type q	Request	Sub-request r	Domain	Acceptance level 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, $d_r = 6, \ldots, \|D\|$	60	5	1
	164–172	164–172	$i_r =$Bob, $d_r = 6, \ldots, \|D\|$	60	5	1
	173–199	173–199	$i_r =$Dan,Eva,Gil, $d_r = 6, \ldots, \|D\|$	60	5	1
6	200–259	200–259	$i_r \in I, d_r \in D : 2 \leq d_r \leq \|D\| - 1$	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

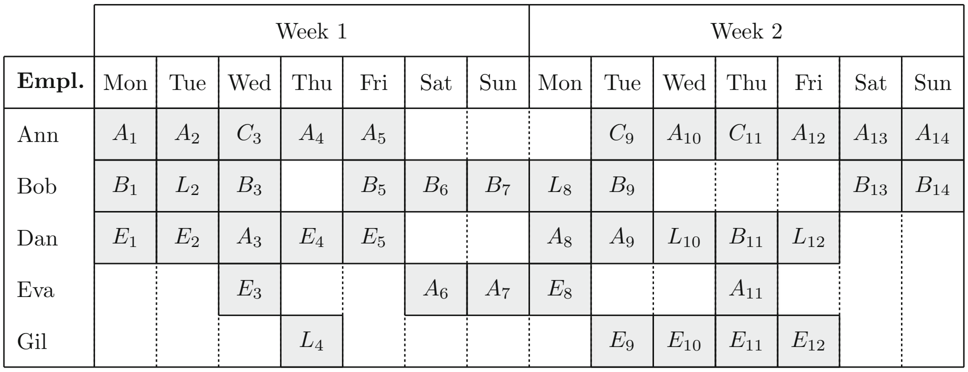


Fig. 6 Optimal schedule

	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}		

critical 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.

Table 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

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.
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

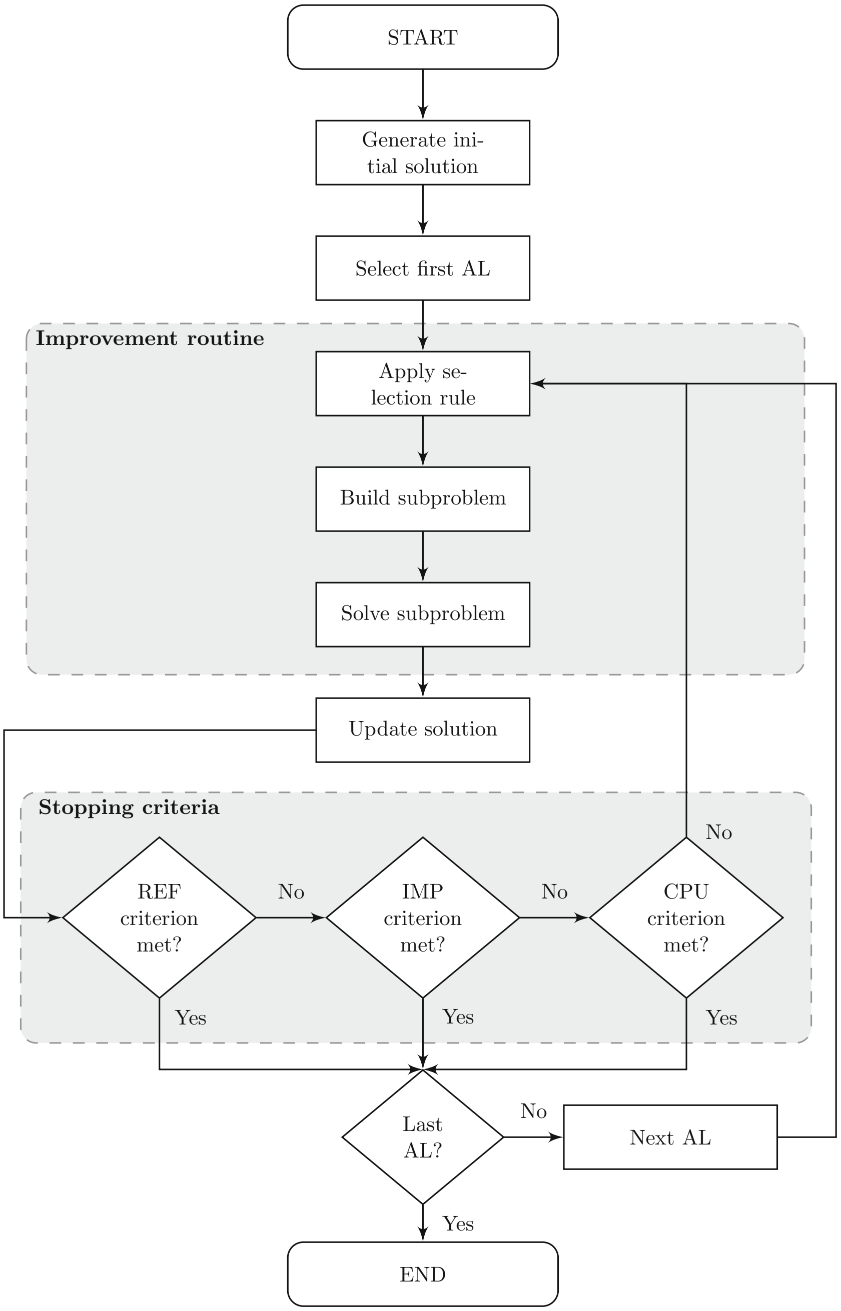
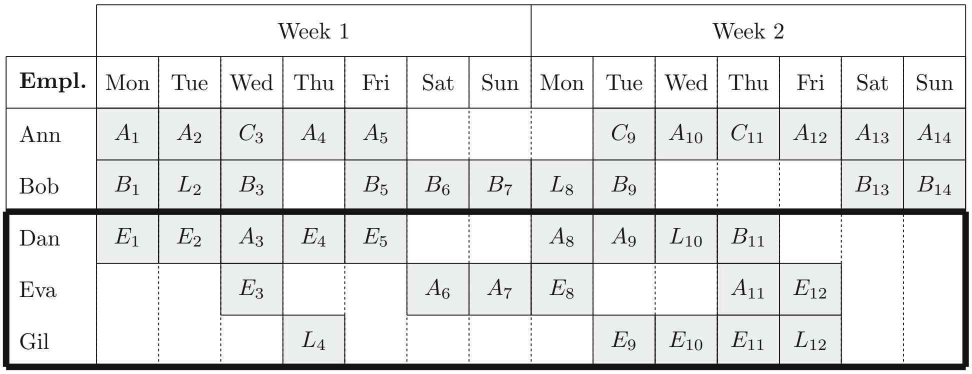


Fig. 7 Flowchart of matheuristic



	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}		

Fig. 8 Temporary schedule in the course of the matheuristic

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 Sect. 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.

5.2 Illustrative example

We applied the proposed matheuristic to the illustrative example introduced in Sect. 2.2. Figure 8 shows a temporary schedule in the course of the matheuristic for an iteration with acceptance level $a^* = 60$. In this schedule, four 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 h

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 Fig. 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 h 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 Fig. 6.

6 Computational analysis

In this section, we evaluate the performance of the proposed approaches. In Sect. 6.1, we present the test instances. In Sect. 6.2, we describe the design of the analysis. In Sect. 6.3, we report and analyze the numerical results.

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 4 weeks. The types of requests to be considered in each instance are those presented in Table 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 h.
- The workload ratio WR: Given the target workloads of employees, the workload ratio determines the number of 8-h 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.
- 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.

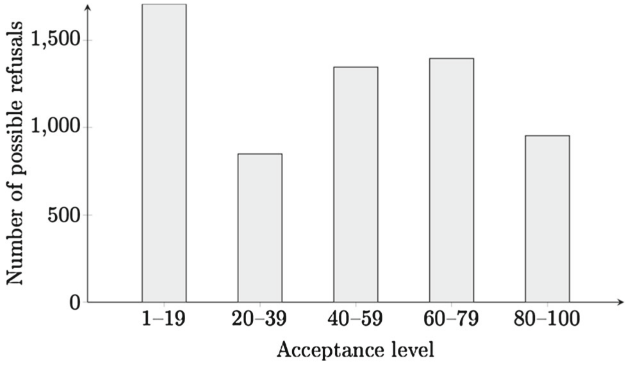


Fig. 9 Average number of possible refusals per acceptance level over all instances

In total, we generated 45 instances, one instance for each possible combination of the three complexity parameters. Figure 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 1, and 23 variants of the request types presented in Table 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").

6.2 Test design

We tested the following approaches:

- *BF*: Basic formulation.
- *AF*: Aggregated formulation.
- *MH*$_k$: Baseline matheuristic as described in Sect. 5 with a subproblem time limit of 3 s. The subscript k indicates the size of the subproblems. We ran MH$_k$ for $k = 4, 5, 6, 7, 8, 9$.
- *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 $k = 4, 5, 6, 7, 8, 9$.
- *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 $k = 8$.
- *MH60*$_k$: Matheuristic MH$_k$ with a subproblem time limit of 60 s instead of 3 s. We ran MH60$_k$ for $k = 8$.
- 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 $k=8$.
- *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 s for the optimization of each individual acceptance level. For all variants of the matheuristics, we used the three stopping criteria presented in Sect. 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 s. The computations were performed on a standard workstation with two 6-core Intel(R) Xeon(R) X5650 2.66 GHz CPUs and 24 GB 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).

6.3 Numerical results

In Sect. 6.3.1, we compare the results of the two exact approaches. In Sect. 6.3.2, we compare these results with the results of the baseline matheuristic. In Sects. 6.3.3–6.3.5, we analyze the results of different variants of the matheuristic and investigate the effectiveness of individual components of the matheuristic. In Sect. 6.3.6, we report the results obtained for the real-world instance.

6.3.1 Exact approaches

Table 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 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 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 Fig. 10. The CPU times are considerably higher for BLPs associated with higher acceptance levels than for BLPs associated

Table 5 Numerical results for exact approaches

		NE					
		Small	Medium		Large		All
		10	30	50	70	90	
OPT	BF	8	2	0	0	0	10
	AF	7	2	1	0	0	10
CPU	BF	269	1075	1367	1671	2025	1281
	AF	244	1095	1345	1495	2073	1250
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

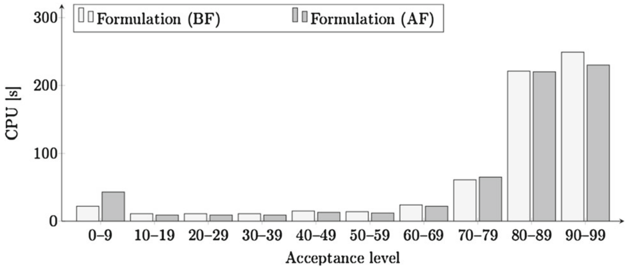


Fig. 10 CPU time per acceptance level

Table 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	1023	1206	1615	1281
	AF	889	1212	1650	1250
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 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	1266	1945	1281
	AF	642	1313	1796	1250
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

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 Fig. 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 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 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 6 and 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 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.

6.3.2 *Matheuristic: comparison with exact approaches*

Table 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 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 8.

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 k. The

Table 8 Comparison of variants of matheuristic

		NE					
		Small	Medium		Large		All
		10	30	50	70	90	
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	1075	1367	1671	2025	1281
	AF	244	1095	1345	1495	2073	1250
	MH$_4$	66	217	686	1363	2049	876
	MH$_5$	128	305	710	1267	1932	868

Table 8 continued

	NE					
	Small	Medium		Large		All
	10	30	50	70	90	
MH_6	207	418	699	1199	1837	872
MH_7	345	471	742	1131	1753	889
MH_8	452	506	757	1160	1692	913
MH_9	479	541	790	1123	1617	910

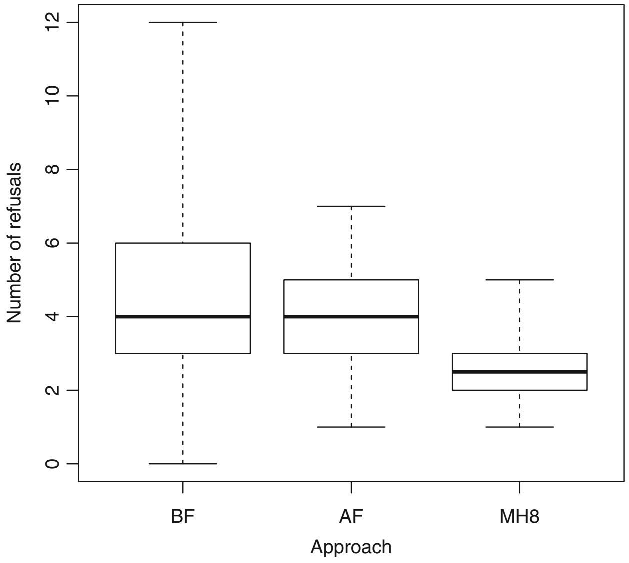


Fig. 11 Distributions of number of refusals among employees for approach BF, AF, and MH_8 for an instance with 90 employees

Table 9 Impact of strategy to formulate the subproblems without redundant constraints and variables

		NE					
		Small	Medium		Large		All
		10	30	50	70	90	
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	1160	1692	913
	MHF_8	437	595	1047	1782	3260	1424

impact of k is analyzed based on the results given in Table 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 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 in terms of both solution quality and CPU time requirement. The outperformance is most distinct for medium- and large-sized instances.

6.3.4 Matheuristic: impact of the employee selection rule

In this section, we examine the impact of the employee selection rule. In Table 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

Table 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	1403	1172	1820
5	35	13	868	1339	967	1425
6	33	18	872	1267	795	1163
7	32	20	889	1253	700	977
8	34	20	913	1228	609	853
9	31	26	910	1187	573	776

Table 11 Comparison of different variants of the employee selection rule

		NE					
		Small	Medium		Large		All
		10	30	50	70	90	
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	1160	1692	913
	MH_8^{2-2}	458	493	723	1049	1518	848
	MH_8^{1-2}	444	521	780	1159	1698	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

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 in terms of both 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 10).

We also investigated other specific employee selection rules. In Table 11, we compare 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.

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 12, we compare the results of MH_8 (with a default subproblem time limit of 3 s) with the results of approach $\mathrm{MH60}_8$ which uses a subproblem time limit of 60 s. Interestingly, increasing the subproblem time limit to 60 s 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.

Table 12 Impact of subproblem time limit

		NE					
		Small	Medium		Large		All
		10	30	50	70	90	
OPT	MH_8	7	3	1	0	0	11
	$\mathrm{MH60}_8$	6	2	0	0	0	8
NBE	MH_8	7	8	9	8	7	39
	$\mathrm{MH60}_8$	8	3	1	1	2	15
ARE	MH_8	2.62	2.29	2.32	2.45	2.54	2.45
	$\mathrm{MH60}_8$	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
	$\mathrm{MH60}_8$	2.71	1.55	1.97	4.30	5.16	3.14
CPU	MH_8	452	506	757	1160	1692	913
	$\mathrm{MH60}_8$	496	585	815	1184	1690	954
NSP	MH_8	454	558	710	703	620	609
	$\mathrm{MH60}_8$	368	482	596	658	600	541

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 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 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.

Table 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	4482	71	2061	2357	68

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.

References

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 and 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 and 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 and 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, Extra Volume: Optimization Stories*, 107–121.

Boschetti, M. A., Maniezzo, V., Roffilli, M., & Bolufé Röhler, A. (2009). Matheuristics: Optimization, simulation and control. In M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, & A. Schaerf (Eds.), *Hybrid metaheuristics: 6th international workshop on hybrid metaheuristics* (pp. 171–177). Heidelberg: Springer.

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 and 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* (pp. 11–22). Boston, MA: Springer US. doi:10.1007/978-1-4419-5771-9_2.

Jones, D., & Tamiz, M. (2016). A review of goal programming. In S. Greco, M. Ehrgott, & J. R. Figueira (Eds.), *Multiple criteria decision analysis: State of the art surveys* (pp. 903–926). New York: Springer.

Jones, D. F., & Tamiz, M. (2002). Goal programming in the period 1990–2000. In M. Ehrgott & X. Gandibleux (Eds.), *Multiple criteria optimization—State of the art annotated bibliographic surveys*. Dordrecht: Kluwer Academic Publishers.

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.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., 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 M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, & L. A. Wolsey (Eds.), *50 years* *of integer programming 1958–2008: From the early years to the state-of-the-art* (pp. 619–645). Heidelberg: Springer.

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*. New York: Springer.

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 C. Blum, M. J. B. Aguilera, A. Roli, & M. Sampels (Eds.), *Hybrid metaheuristics: An emerging approach to optimization* (pp. 31–62). Heidelberg: Springer.

Rihm, T., & Baumann, P. (2015a). Improving fairness in staff assignment: An approach for lexicographic goal programming. In T. Magnanti, K. Chai, R. Jiao, S. Chen, & M. Xie (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 Z. Hanzálek, G. Kendall, B. McCollum, & P. Šůcha (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*. Oxford: Pergamon Press.

Smet, P., & Vanden Berghe, G. (2012). A matheuristic approach to the shift minimisation personnel task scheduling problem. In D. Kjenstad, A. Riise, T. E. Nordlander, B. McCollum, & Burke (Eds.). *Proceedings of the 9th international conference on the practice and theory of automated timetabling, Son* (pp. 145–160).

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., 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 and 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). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*, 367–385.