Ann Oper Res
DOI 10.1007/s10479-017-2546-8

Simulated annealing approach to nurse rostering benchmark and real-world instances

Frederik Knust[2] · **Lin Xie**[1]

Abstract The nurse rostering problem, which addresses the task of assigning a given set of activities to nurses without violating any complex rules, has been studied extensively in the last 40 years. However, in a lot of hospitals the schedules are still created manually, as most of the research has not produced methods and software suitable for a practical application. This paper introduces a novel, flexible problem model, which can be categorized as ASBN|RVNTO|PLG. Two solution methods are implemented, including a MIP model to compute good bounds for the test instances and a heuristic method using the simulated annealing algorithm for practical use. Both methods are tested on the available benchmark instances and on the real-world data. The mathematical model and solution methods are integrated into a state-of-the-art duty rostering software, which is primarily used in Germany and Austria.

Keywords Nurse rostering problem · Flexible model · $\alpha|\beta|\gamma$ notation · Simulated annealing · Mixed integer programming · Real-world data · Duty rostering software

1 Introduction

The healthcare sector is growing rapidly: according to BDI (2013) its share of Germany's GDP will rise from 10% in 2008 to approximately 15% in 2020. Also, it faces a lot of challenges, such as a constant shortage of qualified nurses, which is predicted to become even worse in the future (Kellogg and Walczak 2007; Online Z 2013). Therefore, one way to diminish this shortage is to reduce the time spend by the nurses for secondary tasks, such as nurse rostering. The nurse rostering problem (in short: NRP) is a special case of the staff scheduling problem, which deals with assigning nurses to a given set of activities for a given

✉ Lin Xie
xie@leuphana.de

1 Leuphana University of Lüneburg, Scharnhorststr. 1, 21335 Lüneburg, Germany

2 Connext Communication GmbH, Balhorner Feld 11, 33106 Paderborn, Germany

time period in a hospital. The generated schedule for each nurse is called in this paper a *roster*. Although this task of nurse rostering does not directly involve taking care of the patients, it is still important. First of all, the generated rosters need to ensure that enough nurses are on duty at all times to provide appropriate care for the patients (Wright et al. 2006). Additionally, the quality of the rosters has a major influence on the morale of the nurses (Drake 2014). The measure for the quality includes fairness among nurses (such as overtime) and the nurses' individual preferences (Kellogg and Walczak 2007; Drake 2014). Ignoring these factors will have a negative impact on the nurses' morale, and may even cause some nurses to call in sick if their requests are not satisfied (Drake 2014).

Although an extensive amount of research has been invested into methods to automate the NRP, many hospitals and wards still create their rosters manually (Burke et al. 2004b; Kellogg and Walczak 2007). This task can take up to several days to complete (Burke et al. 2006). According to Kellogg and Walczak (2007) there are severals reasons to explain that. First, some researchers do not intend to implement their model for a practical use. Second, many nurses have the impression that they lack the time to learn a new software system. Third, the software systems developed in the context of academic research often do not provide a sufficient amount of customer support, although it is important to provide such support. The reason for this is setting up the parameters for the automatic scheduling is often rather complex. Lastly, most of the methods do not support self-scheduling, which is a technique often used in practice. Furthermore, Drake (2014) concludes that the parameters which apply to the scheduling are almost unique for each hospital or even ward, narrowing the possibility to apply a scheduling method to a broader range of hospitals. Drake's findings are supported by Kellogg and Walczak (2007), who found out that half of the practically applied methods are only used in a single hospital.

Moreover, the NRP has been proved to be NP-hard (Osogami and Imai 2000). As with all staff scheduling problems, the NRP is highly constrained, with the difference that hospitals must be staffed around the clock and the margin of error is extremely low (Ernst et al. 2004), as, unlike for different areas of staff scheduling, undercoverage in a hospital due to misplanning endangers the well-being or even the life of the patients. The staffing problems become a challenge, because a lot of rules (e.g., the law, individual and labor agreements) and the qualifications of the nurses must be considered. Therefore, it is crucial to create a good and feasible schedule, which ensures adequate staff coverage at all times. The use of network flow formulations for the rules was proved to be effective on a benchmark dataset in Smet et al. (2014). This is supported by Xie and Suhl (2015), who showed the benefits of using network structure to formulated the complex rules while condering the qualification of the employees. This paper uses the network design, called multi-commodity flow network, which was inspired by the network designs of Xie and Suhl (2015) and Cappanera and Gallo (2004) for solving the staff scheduling problems in transportation sectors. The similar network design is used in Burke and Curtois (2014) for solving the NRP, but they used a network layer individually for each nurse to solve the pricing problem in their Branch and Price method. Instead, we use the network design to formulate the whole problem. Smet et al. (2014) use different network design, which the nodes are defined differently to our work, namely the nodes represent shifts, times, nurses. More details about our network design can be found in Sect. 3. A short list of publications about the applications of network flows for the NRP can be found in Smet et al. (2014).

The aim of this paper is to introduce an automated method to create rosters, which addresses the issues preventing a practical application. For this purpose, a novel, flexible problem model was developed, which can be used to model the varying parameters of diverse hospitals and wards. This problem is classified as ASBN|RVNTO|PLG according to the

$\alpha|\beta|\gamma$ classification of Causmaecker and Vanden Berghe (2010). A robust solution approach is presented, which creates good schedules for different problem instances in a short amount of time. Additionally, the method is integrated into a state-of-the-art duty rostering software, which is already in use in many hospitals especially in Germany and Austria. Hence, the additional training required by the nurses is kept low and customer support can be provided when necessary. In addition, the software features self-service functionality, which allows nurses to enter individual preferences or even plan their whole schedule. These preferences are considered by our solution approach as far as possible.

The first part of the paper gives a brief introduction to the nurse rostering problem and a literature review (see Sect. 2). Next, the network design will be described in Sect. 3. After that, we describe an exact and a heuristical solution approach that we implement and their results in Sects. 4 and 5. Finally, we provide a summary in the last section.

2 Nurse rostering problem

In this section, we first give details about the input information for solving the NRP. Then, a brief overview on the state of the art is given, with the focus on the approaches, which were applied successfully to the real-world problems.

2.1 Input information

The nurse rostering problem aims at assigning activities to nurses for a given time period (e.g., a couple of weeks or months), while respecting a number of rules and regulations. Therefore, the input information for this problem includes activity, nurse information, and rules and regulations. Note that the roster from the previous period is also considered in the nurse rostering problem, since some restrictions about rules and regulations should be checked in the overlap with previous and actual planning periods. For example, we consider a restriction which limits the amount of work-related activities for each nurse per week. As the planning period is flexible, it is possible that the first day of the actual planning period falls in an ongoing week. In order to check this restriction, at least the missing part of the week in the previous period should be considered. However, the assigned activities from the previous period are fixed and cannot be changed during the optimization. The previous period should be at least as long as the maximum length among all rules. In the test cases of this paper, 15 days from the previous period are usually included.

2.1.1 Activity

Each *shift* has a *shift type*, which depends on its start and end times, e.g., an early shift begins at 7am and ends at 12 noon. The working time of a shift (type) is equal to the ending time minus the beginning time, including breaks. The number of shifts in this work is variable (the category $\beta : N$ of $\alpha|\beta|\gamma$ notation of Causmaecker and Vanden Berghe 2010); moreover, the definition of different shift types is not fully separated in time (see the example on the left-hand side of Table 1; the category $\beta : O$). Not only shifts are assigned to nurses, but also some other activities, including standbys, days off and planned absences (e.g., vacations and training periods; the category $\alpha : A : a$). *Standby* activities are planned to cover the absences of nurses, while *days off* consist of a couple of rest days between working days. The *planned absences* are fixed for each nurse and cannot be changed in the optimization.

2.1.2 Nurse information

The nurse information includes not only his/her level of qualification and experience (head nurse, regular nurse, junior nurse), but also his/her contract (regular nurse, part-time nurse and so on; the category $\alpha : A : b$) and work account. The nurse can only carry out the jobs that he/she is qualified to do. The contractual weekly working times and the number of vacation days of each nurse can be found in his/her contract (the category $\alpha : A : d$), while his/her work account includes the current overtime and the number of days off from the previous periods. A nurse with more overtime in the previous periods can get more jobs with shorter working hours in order to reduce overtime gradually. Moreover, each nurse can express his/her preferences, including his/her daily desired activity and the possible combination of activities (the category $\alpha : A : e$). For example, a nurse wishes to get one activity on 1 day (e.g., a day off), or does not wish to get an undesired shift (e.g., a late-night shift), or does not wish to get an isolated day off between 2 working days.

2.1.3 Rules and regulations

We consider two types of rules: horizontal and vertical rules. Horizontal rules depend only on one individual roster, while vertical rules combine the information among all rosters.

Horizontal rules These rules consist of compatibility, working block, preferences of nurses and fairness among all nurses.

The *incompatible* connections (the category $\alpha : S : c, d$) are gathered in a list of forbidden sequences, which violate contractual or legal resting time, for example, a night shift followed by an early shift is forbidden due to the short rest time between them. It is possible that a combination of activities, which are important for the quality of the solution, is regarded as a forbidden sequence as well. For example, complete weekends are considered in many publications (e.g., in Cheang et al. 2003; Burke et al. 2004a; Maenhout and Vanhoucke 2009; He and Qu 2012; Lü and Hao 2012). Therefore, the combinations of activities causing mixed weekends are forbidden, where a nurse is working on 1 day of the weekend and has the other day off. According to Burke et al. (2008), it is possible to define which days are part of the weekend specifically: for example, the weekend can go from Saturday to Sunday, from Friday to Sunday, or from Friday to Monday. The length of all forbidden sequences is in this work limited to two, therefore the forbidden sequences can be implicitly considered during the network generation (see Sect. 3). The maximum consecutive number of working days for a full-time nurse is restricted to, for example, five (*working block*, the category $\alpha : S : a$).

As mentioned before, the *preferences of nurses* include the daily desired activities and the possible combination of activities. Undesirable combinations of activities include, for example, an isolated working day between days off, a day off between 2 working days, and dissimilarities of shifts within a given number of days. The nurses desire to get shifts that are as similar as possible, for example, within a working week. The means having the same or a similar shift type, such as an early shift. The *fairness* among all nurses (the category $\alpha : B$) includes fair distribution of undesirable shift types (such as night shifts) and a balanced workload. There are different ways to model the balanced workload. First, as shown in Burke and Curtois (2014), the minimum and maximum number of working days will be counted, for example, a nurse with a full-time contract must work at least four and at most 5 days each week, while a nurse with a part-time contract must work at least two and at most 3 days each week. However, this approach is only applicable if all shift types have a similar length. Due to the limited flexibility, the weekly working hours of each nurse are considered in this paper

Table 1 Shift types (top) with corresponding shift groups (bottom)

Name	Abbr.	Start	End
Early shift 1	E1	06:00	14:30
Early shift 2	E2	07:00	11:00
Noon shift	NO	10:30	16:30
Late shift 1	L1	13:15	21:00
Late shift 2	L2	14:00	22:30
Night shift	N	22:00	06:30

Timeframe	Shift group	
	#	Shift types
06:00–06:30	I	N, E1
06:30–07:00	II	E1
07:00–10:30	III	E1, E2
10:30–11:00	IV	E1, E2, NO
11:00–13:15	V	E1, NO
13:15–14:00	VI	E1, NO, L1
14:00–14:30	VII	E1, NO, L1, L2
14:30–16:30	VIII	NO, L1, L2
16:30–21:00	IX	L1, L2
21:00–22:00	X	L2
22:00–22:30	XI	L2, N
22:30–06:00	XII	N

instead. That means the minimum and maximum of the nurse's weekly working hours are considered. Moreover, the minimum amount of days should be considered as well, especially for the case of a short and split week at the end of the planning period. If we get a split week and its amount is larger than the minimum amount of days, then the expected working hours are reduced accordingly. Moreover, the overtime/shortfall of each nurse and the maximum overtime/shortfall of all nurses are penalized. Additionally, a percentage deviation (such as 10%) of the minimum and maximum working hours is allowed.

Vertical rules These rules (the category β: R, T, V) ensure that the coverage demand is fulfilled at all times, which includes shift-based demand (Ernst et al. 2004; Burke et al. 2004b, 2010) and time-based demand (Vanden Berghe 2002). *Shift-based demand* specifies how many qualified nurses shall be assigned minimally and maximally to each shift, while *time-based demand* defines the number of nurses who must be present at all times.

The time-based demand is more complex than the shift-based demand, since nurses are assigned to shifts, with predefined time periods, but not to arbitrary time slots. Therefore, the task is to find a combination of shift types which can fulfill the time-based demand as exactly as possible. To achieve this, the shift types are combined into shift type groups, while each shift type group covers a disjunct time period. Table 1 shows how shift type groups are created for a given set of shift types.

One problem that occurs for dealing with the real-world problem is that the number of shift type groups increases significantly, since more shift types (such as ≥ 15) are considered. Moreover, the matching shift type groups should be generated for each shift on all days of the planning period. Also, there are different time-based coverage demands to address different qualification requirements. One possible way to reduce the number of groups is provided by

Vanden Berghe (2002): groups that are completely covered by another group are ignored. For example, in Table 1, the groups III to VII can be removed, because they all contain the shift type *E1* and there is a group (II) containing only *E1*. There is also the scenario where a nurse is assigned to a shift that causes coverage to serveral groups. For example, if a nurse is assigned to an *Early Shift 1*, the coverage of groups II to VII is increased. As the timeframes of the shift types are overlapping, there are consequently shift groups, such as group VII, which are overstaffed. The overstaffing is often prohibited, except the case of the overlapping (e.g., 15 min) at the time of the team switch for a short briefing (Burke et al. 2006; Causmaecker and Vanden Berghe 2010).

The presented approach is more flexible than the one used by Burke et al. (2006). They handle the time-based coverage demands by searching for sets of shift types which fulfill the coverage demands as closely as possible. The set leading to the lowest penalty costs is then used in the roster. However, their approach depends on the combination of shift types, which are fully separated in time. If the shift types are not defined that way, their approach might not find all shift type combinations providing the desired amount of coverage.

2.2 State of the art

The NRP has been researched extensively since the early 1970s, mostly with the aim to create a feasible (and good) schedule automatically. Two different ways are mainly used by the researchers to create a roster, namely cyclic and non-cyclic rostering (see more details in Xie and Suhl 2015 for the similar rostering problem in public bus transit).

Cyclic rostering aims at determining a set of cyclic rosters, which satisfy the different rules and regulations. These rosters can be created for different time periods (e.g., 2 weeks), after which they are repeated. To complete the rostering, each nurse is allocated to one cyclic roster. This approach has several advantages, e.g., the nurses can plan their free-time activities a long time ahead due to the repeated rosters; moreover, low computational effort is expected (Burke et al. 2004b). This way of generation also has disadvantages, such as a lack of flexibility, which is why Burke et al. (2004b) conclude that cyclic scheduling can only be used in rare cases. Maenhout and Vanhoucke (2009) come to the same conclusion with the argument that considering the preferences of the nurses in the rostering problem has a major influence on the quality of the healthcare service. In their opinion the motivation of the nurses is improved if their individual working preferences are considered by the rosters. Cyclic rostering is less likely to match these preferences (see also Xie and Suhl 2015 for the same opinion).

Following this perception, most of the recent contributions to the NRP focus on non-cyclic (also individual or personalized) scheduling. This approach is advantageous, as the individual preferences of the nurses, planned absences (e.g., vacations and trainings) and individual contractual regulations (e.g., no night shifts) can be taken into account, making the schedule more flexible. On the downside, the NRP for generating an individual roster for each nurse increases the complexity of the problem; therefore, it is harder to solve (Cheang et al. 2003; Xie and Suhl 2015).

Many different solution approaches are proposed in the literature, such as mathematical programming (MP), (meta-)heuristics, constraint programming and hybrid approaches. Pure MP approaches are proven to be inappropriate for practical applications of the NRP, due to the large amount of hard and soft constraints (Lim et al. 2012). Simplifying the models leads to solutions which are not applicable to real-world scenarios. However, without changing the models, the computational times were too high for practical use. Therefore, hybrid approaches including combining integer programming with a *variable neighborhood search* metaheuristic are applied successfully (Burke et al. 2010; Lim et al. 2012; Valouxis et al. 2012).

Among the metaheuristics, especially simulated annealing (Bai et al. 2010; Hadwan and Ayob 2010), tabu search (Burke et al. 1998; Dowsland 1998) and genetic algorithms (Aickelin and Dowsland 2008; Bai et al. 2010), are used to address the NRP with alternating results. Recently, the algorithm, variable neighborhood search, receives some attention (Burke et al. 2008, 2010; Qu and He 2010). The more successful contributions are those that use hybrid or two-phase approaches to address the shortcomings of the different metaheuristics, e.g., it is necessary to alter the parameters of heuristics to achieve good results for different problem instances (Burke et al. 2001). This instance-dependent parameter setting is a severe problem, as the creator of the roster usually does not have sufficient knowledge to alter the parameters to fit the local environment. To address this issue, parameter-independent metaheuristics or adaptive approaches based on machine learning techniques, which allow automated adjustment of the parameters (Bai et al. 2010), could instead be applied in practice.

Unfortunately, most of the contributions in the literature cannot be applied to real-world problems, because the models are either too simple or not flexible enough (or both) (Burke et al. 2004b). There are only a few contributions which focus on defining a flexible problem model. Most contributions use a fixed set of hard and soft constraints, a small number of predefined shift types and shift-based coverage. Causmaecker and Vanden Berghe (2010) introduce an $\alpha|\beta|\gamma$ notation to categorize different approaches to the NRP. They also provide a classification for 25 literature sources from 2003 to 2010 and show that only ten of these sources allow a flexible amount of shift types. Four of those ten allow shift types with overlapping time and a flexible shift coverage demand (i.e., it is possible to define different coverage demands for each day). Only one contribution remains which also allows time-based coverage demand (Burke et al. 2006), which uses a model called advanced nurse rostering model (ANROM) introduced in Vanden Berghe (2002). Vanden Berghe (2002) was probably the first one who realized that a model allowing flexible shift type definitions is necessary to produce schedules which can be used in real-world situations. This observation was based on feedback provided by the users of *Plane*, a nurse rostering software used in Belgian hospitals (Vanden Berghe 2002, p. 29). The model includes lots of different constraints: soft constraints to increase the quality of the schedule for the nurses (e.g., ensuring minimum times between two assignments) and hard constraints for the coverage and skill requirements (Vanden Berghe 2002, pp. 35, 40). The model also supports time-based coverage definitions (also called *floating personnel requirements* by the author). The coverage definitions are translated into all shift type combinations which provide the required coverage (Vanden Berghe 2002, pp. 172ff.). The shift type combination resulting in the lowest costs is used in the schedule. The model includes a lot of predefined constraints. It does not, however, allow changing the type of a constraint (hard or soft) or to model arbitrary constraints. For example, it is possible to define a maximum number of shift types in a week, but not a minimum, and it is not possible to define undesired shift sequences besides those which have been modeled explicitly (e.g., the maximum number of consecutive (work) days (Vanden Berghe 2002, p. 42).

However, this is possible in the problem model introduced by Burke and Curtois (2014). They use a pattern-based approach which can be used to model different kinds of constraints. For example, to model the requirement that a night shift must always be followed by another night shift or a day off, the following pattern would be used: “maximum zero matches of the pattern ‘N followed by any shift other than N”’ (Burke and Curtois 2014). It is possible to provide a *quantifier*, to specify how often a pattern shall be matched (minimum and maximum). A constraint can be restricted to only apply to a certain part of the planning period. In addition to the pattern-based constraints, they introduced a *workload* constraint to specify the minimum and maximum amount of working times for a nurse within a time period. Another interesting problem model is used during the first international nurse rostering competition in 2010 (Haspeslagh et al. 2012). They use a hybrid approach, predefined constraints (e.g., to specify a minimum and maximum amount of shifts and the maximum amount of consecutive work days) and pattern-based constraints to model undesired shift sequences. However, time-based coverage demands and a flexible definition of soft and hard constraints are not supported. Some other authors published flexible models as well, but did not provide the same amount of flexibility as Burke et al. (2004b), Burke and Curtois (2014) (e.g., the model introduced by Bilgin et al. 2012 does not support time-based coverage).

3 Network design

As mentioned before, the network design we used in this work is based on the network design of Xie and Suhl (2015) and Cappanera and Gallo (2004) in transportation sectors, namely the multi-commodity flow network, where the activities are represented as nodes and they are connected by arcs with the activities from the previous and the following days. The compatible arcs are generated for each nurse, i.e., the arcs are identified by the nurses; so we get one single network for all nurses (main difference to Xie and Suhl 2015 and Cappanera and Gallo 2004, where a network layer is generated for each employee). The compatibility means that the generated arcs do not violate any work regulations, such as rest time between 2 working days; moreover, two activities connected by an arc are also feasible if the arc is generated for a nurse. For example, an arc connecting at least one night shift is not allowed to be generated for a nurse who does not work at night due to the contract. Therefore, generating a roster for a nurse a is equal to sending a single unit of flow from an activity on the first day to an activity on the last day. The generated roster should follow the horizontal rule defined individually for that nurse a. As mentioned before, the daily desired activities and the possible combination of activities are considered for each nurse (preferences of nurses). We can formulate them in the network as costs, i.e., the cost of each connection arc is used to reflect the combination of activities and the cost of used source and target activities of the arc. Figure 1a depicts the basic structure of the network for four activities (from top to bottom: early shift, late shift, night shift and day off) and the arcs here are associated with one single nurse. As mentioned before, some horizontal rules are considered during the generation of the network, including:

1. The arcs of forbidden sequences (with length two) are not generated, including those violating contractual or legal rest time, or complete weekends. For example, a reduced network can be found in Fig. 1b, where we assume that the weekend starts as the late and night shifts on Friday, and ends as the early shifts on Monday. Therefore, the arcs connecting between the late/night shift on Friday and day off on Saturday can be removed. The same holds for the rest of the weekends.
2. Undesired combinations of activities, for example, all night shifts, are not desired. In such cases, either the arcs connecting night shifts between each 2 consecutive days (such as days 1 and 2, days 2 and 3) are removed, or the cost of such an arc is increased (see left-hand side and right-hand side of Fig. 1c, respectively). In this example, the arc with the color red receives higher costs than the ones with the color orange. The other undesired connection arcs of activities can be handled similarly, such as an isolated working day/day off between days off/working days, or dissimilarities of shifts.
3. The activities from the previous period or the preassigned activities (such as vacations) cannot be changed; therefore, if a fixed activity exists on 1 day, then the incoming and outgoing arcs connected with the other activities on that day are removed (see Fig. 1d: the

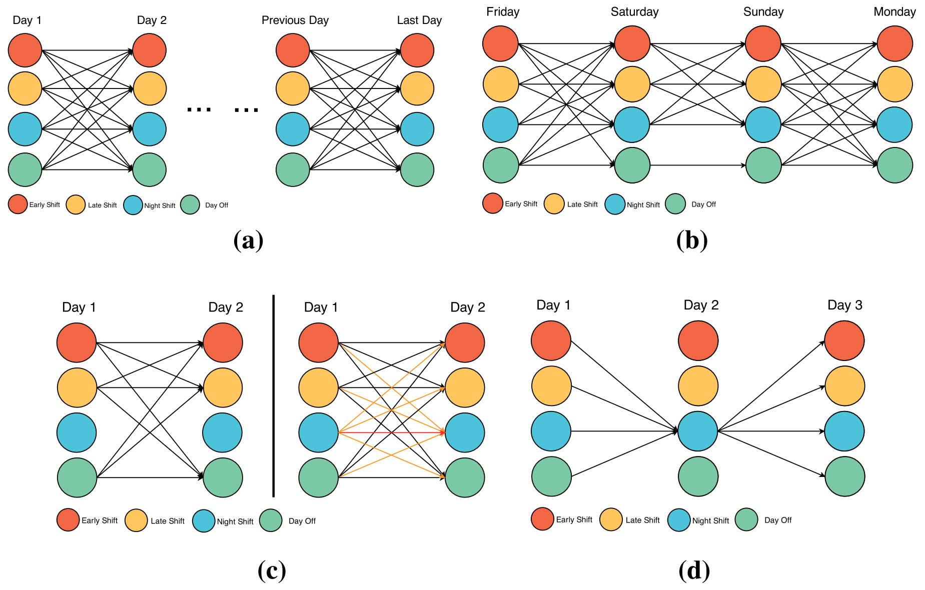

Fig. 1 An example of the network structure. **a** Basic network structure. **b** The network structure after considering complete weekends. **c** The network structure after considering undesired combinations of activities. **d** The network structure after considering fixed activities

night shift is fixed for the nurse on day 2). Also, there is one connection arc between two fixed activities. Note that the work regulations are checked from the previous period to the actual period, e.g., if the maximum number of consecutive working days is achieved at the end of the last period, then this period begins with days off. In such a case, the arcs connected to the work-related activities at the beginning of this period are removed.

This network design is used both for the exact approach in Sect. 4 and the simulated annealing algorithm in Sect. 5.

4 Exact approach

Based on the network design in the previous section, the mathematical MIP model will be described in Sect. 4.1. Thus, the results produced by different MIP solvers will be shown in Sect. 4.2.

4.1 Mathematical model

The MIP model is shown in this section, which includes the basic and specialized index-sets, all necessary parameters and variables, and the constraints and the objective function.

Index-sets

$\mathcal{N}$	Set of all nurses n.
$\mathcal{D}$	Ordered set of all days d in the optimization period.
$\mathcal{W}$	Ordered set of all weeks w in the optimization period.
$\mathcal{D}_w \subseteq \mathcal{D}$	Ordered set of all days d in the week w.

$\mathcal{S}$	Set of all activities s.
$\mathcal{S}^W \subset \mathcal{S}$	Set of all work-related activities.
$\mathcal{S}_d \subset \mathcal{S}$	Set of all activities on day d.
$\mathcal{S}_w \subset \mathcal{S}$	Set of all activities in week w.
$\mathcal{A}$	Set of all arcs a.
$\mathcal{A}_n \subseteq \mathcal{A}$	Set of all arcs associated to a nurse n.
$\mathcal{F}_{s,n} \subset \mathcal{A}_n$	Set of all arcs outgoing from activity s for a nurse n (Forward-Star).
$\mathcal{B}_{s,n} \subset \mathcal{A}_n$	Set of all arcs incoming to activity s for a nurse n (Backward-Star).
$\mathcal{R}$	Set of all rules r which apply to the optimization problem.
$\mathcal{R}^{\text{Soft}} \subseteq \mathcal{R}$	Set of all rules r which are formulated as soft constraints in the optimization problem.
$\mathcal{R}^{\text{Hard}} \subseteq \mathcal{R}$	Set of all rules r which are formulated as hard constraints in the optimization problem.
$\mathcal{R}^C \subseteq \mathcal{R}$	Set of all rules of *shift-based demand*.
$\mathcal{R}^G \subseteq \mathcal{R}$	Set of all rules of *shift type group*.
$\mathcal{R}^{GC} \subseteq \mathcal{R}$	Set of all rules of *shift type group coverage*.
$\mathcal{R}^S \subseteq \mathcal{R}$	Set of all rules of *activities combination*.
$\mathcal{R}^T \subseteq \mathcal{R}$	Set of all rules of *bounds of activities*.
$\mathcal{R}^W \subseteq \mathcal{R}$	Set of all *working time* rules, such as minimum/maximum working hours a day, per week, and per month.
$\mathcal{R}^H \subseteq \mathcal{R}^W$	Set of all *weekly working hours* rules.

Parameters

L_r^{Min}	Lower bound of the rule r.
L_r^{Max}	Upper bound of the rule r.
C_a^A	Costs which apply if an arc a is used.
C_r^R	Costs which apply for each violation of the rule r.
N_r	Nurses who are affected by the rule r.
S_r	Activities which are affected by the rule r.
$S_{r,d}$	Activities which match the activities combination rule r starting on day d.
G_r	The *shift group* g of a shift type group or shift type group coverage rule r.
D^F	The first day of the optimization period.
D_g	The number of days covered by a shift type group g.
D_r^{Sequence}	The number of days covered by a rule of activities combination r.
H_s	The working time (in hours) of an activity s.
$W_{w,n}$	The weekly (contractual) working time (in hours) of nurse n in week w. The working time is dependent on the week, because in incomplete weeks (at the beginning or end of the optimization period) only a fraction of the regular working time is used (e.g., if 5 days of a week fall into the optimization period, only five-sevenths of the regular working time is used as expected working time for that week).
A_r	The allowed (unpenalized) deviation from the weekly working time rule r (in percent).
U_r	The costs ratio of a weekly working time rule r between the overtime and shortfall costs. A value < 1 causes shortfall to be penalized less than overtime and vice versa. A value of 1 causes shortfall and overtime to be treated equally.
T_r	The costs ratio of a weekly working time rule r between the individual costs for each nurse's overtime and the weekly maximum overtime. Usually, this

value is > 1, as the maximum overtime should be penalized more than the individual one.

Variables

x_a	Binary variable that depicts the flow over the arc a.
$y_{s,n}$	Binary variable that indicates whether a nurse n has an activity s.
$c_{r,s}^{\text{Min}}$	Integer variable storing the lower bound of the number of violations of the rule of shift-based demand r for the activity s.
$c_{r,s}^{\text{Max}}$	Integer variable storing the upper bound violations of the rule of shift-based demand r for the activity s.
$c_{r,g}^{\text{Group,Min}}$	Integer variable storing the lower bound violations of the rule of shift type group coverage r for the shift group g.
$c_{r,g}^{\text{Group,Max}}$	Integer variable storing the upper bound violations of the rule shift type group coverage r for the shift g.
$g_{r,g,n}^{\text{Group}}$	Binary variable which is set to one iff the shift group g from the rule r is active for nurse n.
$g_{r,n}^{\text{Min}}$	Integer variable storing the lower bound violations of the rule of shift type group r for nurse n.
$g_{r,n}^{\text{Max}}$	Integer variable storing the upper bound violations of the rule of shift type group r for nurse n.
$\omega_{r,n}^{\text{Min}}$	Integer variable storing the lower bound violations of the rule of activities combination r for nurse n.
$\omega_{r,n}^{\text{Max}}$	Integer variable storing the upper bound violations of the rule of activities combination r for nurse n.
$h_{r,n}^{\text{Min}}$	Linear variable storing the lower bound violations of the working time rule r for nurse n.
$h_{r,n}^{\text{Max}}$	Linear variable storing the upper bound violations of the working time rule r for nurse n.
$v_{r,d,n}$	Binary variable indicating whether the rule of activities combination r has been violated on the starting day d for nurse n.
$e_{r,w,n}^{+}$	Linear variable storing the overtime according to the rule of weekly working hours r in week w for nurse n.
$e_{r,w,n}^{-}$	Linear variable storing the shortfall according to the rule of weekly working hours r in week w for nurse n.
$e_{r,w,n}^{\text{Week},+}$	Linear variable storing the maximum overtime according to the rule of weekly working hours r in week w.
$e_{r,w,n}^{\text{Week},-}$	Linear variable storing the maximum shortfall according to the the rule of weekly working hours r in week w.

4.1.1 Hard constraints

$$y_{s,n} = \sum_{a \in \mathcal{B}_{s,n}} x_a \qquad \forall n \in \mathcal{N}, \forall d \in \mathcal{D} \setminus \left\{ D^F \right\}, \forall s \in \mathcal{S}_d$$

$$y_{s,n} = \sum_{a \in \mathcal{F}_{s,n}} x_a \qquad \forall n \in \mathcal{N}, \forall d \in \mathcal{D} \setminus \left\{ D^L \right\}, \forall s \in \mathcal{S}_d \tag{1}$$

$$\sum_{s \in \mathcal{S}_d} y_{s,n} = 1 \qquad \forall n \in \mathcal{N}, \forall d \in \mathcal{D} \tag{2}$$

The set of constraints (1) makes sure that the value of a y variable matches the sum of all arcs of the backward-star as well as the sum of all arcs of the forward-star (*flow conservation*), while the set of constraints (2) ensures that exactly one activity is assigned to each nurse per day (*demand satisfaction*).

4.1.2 Mixed constraints

$$\begin{aligned}
&\sum_{n\in N_r} y_{s,n} \geq L_r^{\mathrm{Min}} && \forall r\in\mathcal{R}^{C,\mathrm{Hard}}, \forall s\in S_r\\
&\sum_{n\in N_r} y_{s,n} \geq L_r^{\mathrm{Min}} - c_{r,s}^{\mathrm{Min}} && \forall r\in\mathcal{R}^{C,\mathrm{Soft}}, \forall s\in S_r\\
&\sum_{n\in \mathcal{N}_r} y_{s,n} \leq L_r^{\mathrm{Max}} && \forall r\in\mathcal{R}^{C,\mathrm{Hard}}, \forall s\in S_r\\
&\sum_{n\in \mathcal{N}_r} y_{s,n} \leq L_r^{\mathrm{Max}} + c_{r,s}^{\mathrm{Max}} && \forall r\in\mathcal{R}^{C,\mathrm{Soft}}, \forall s\in S_r
\end{aligned}\tag{3}$$

$$\begin{aligned}
&\sum_{s\in g} y_{s,n} \leq D_g - 1 + g_{r,g,n}^{\mathrm{Group}} * M && \forall r\in\mathcal{R}^{G}, \forall g\in G_r, \forall n\in N_r\\
&\left(1 - g_{r,g,n}^{\mathrm{Group}}\right) * M \geq D_g - \sum_{s\in g} y_{s,n} && \forall r\in\mathcal{R}^{G}, \forall g\in G_r, \forall n\in N_r\\
&\sum_{g\in G_r} g_{r,g,n}^{\mathrm{Group}} \geq L_r^{\mathrm{Min}} && \forall r\in\mathcal{R}^{G,\mathrm{Hard}}, \forall n\in N_r\\
&\sum_{g\in G_r} g_{r,g,n}^{\mathrm{Group}} \geq L_r^{\mathrm{Min}} - g_{r,n}^{\mathrm{Min}} && \forall r\in\mathcal{R}^{G,\mathrm{Soft}}, \forall n\in N_r\\
&\sum_{g\in G_r} g_{r,g,n}^{\mathrm{Group}} \leq L_r^{\mathrm{Max}} && \forall r\in\mathcal{R}^{G,\mathrm{Hard}}, \forall n\in N_r\\
&\sum_{g\in G_r} g_{r,g,n}^{\mathrm{Group}} \leq L_r^{\mathrm{Max}} + g_{r,n}^{\mathrm{Max}} && \forall r\in\mathcal{R}^{G,\mathrm{Soft}}, \forall n\in N_r
\end{aligned}\tag{4}$$

$$\begin{aligned}
&\sum_{s\in g}\sum_{n\in N_r} y_{s,n} \geq L_r^{\mathrm{Min}} && \forall r\in\mathcal{R}^{GC,\mathrm{Hard}}, \forall g\in G_r\\
&\sum_{s\in g}\sum_{n\in N_r} y_{s,n} \geq L_r^{\mathrm{Min}} - c_{r,g}^{\mathrm{Group,Min}} && \forall r\in\mathcal{R}^{GC,\mathrm{Soft}}, \forall g\in G_r\\
&\sum_{s\in g}\sum_{n\in N_r} y_{s,n} \leq L_r^{\mathrm{Max}} && \forall r\in\mathcal{R}^{GC,\mathrm{Hard}}, \forall g\in G_r\\
&\sum_{s\in g}\sum_{n\in N_r} y_{s,n} \leq L_r^{\mathrm{Min}} + c_{r,g}^{\mathrm{Group,Max}} && \forall r\in\mathcal{R}^{GC,\mathrm{Soft}}, \forall g\in G_r
\end{aligned}\tag{5}$$

$$\begin{aligned}
&\sum_{s\in S_{r,d}} y_{s,n} \leq D_r^{\mathrm{Sequence}} - 1 && \begin{array}{l}\forall r\in\mathcal{R}^{S,\mathrm{Hard}}, \forall n\in N_r,\\ \forall d\in\left\{1,\ldots,|\mathcal{D}| - D_r^{\mathrm{Sequence}}\right\}\end{array}\\
&\sum_{s\in S_{r,d}} y_{s,n} \leq D_r^{\mathrm{Sequence}} - 1 + v_{r,d,n} && \begin{array}{l}\forall r\in\mathcal{R}^{S,\mathrm{Soft}}, \forall n\in N_r,\\ \forall d\in\left\{1,\ldots,|\mathcal{D}| - D_r^{\mathrm{Sequence}}\right\}\end{array}
\end{aligned}\tag{6}$$

$$\sum_{s\in S_r} y_{s,n} \geq L_r^{\mathrm{Min}} \qquad \forall r\in\mathcal{R}^{T,\mathrm{Hard}}, \forall n\in N_r$$

$$\sum_{s\in S_r} y_{s,n} \geq L_r^{\text{Min}} - \omega_{r,n}^{\text{Min}} \qquad \forall r \in \mathcal{R}^{T,\text{Soft}}, \forall n \in N_r$$

$$\sum_{s\in S_r} y_{s,n} \leq L_r^{\text{Max}} \qquad \forall r \in \mathcal{R}^{T,\text{Hard}}, \forall n \in N_r$$

$$\sum_{s\in S_r} y_{s,n} \leq L_r^{\text{Max}} + \omega_{r,n}^{\text{Max}} \qquad \forall r \in \mathcal{R}^{T,\text{Soft}}, \forall n \in N_r \tag{7}$$

$$\sum_{s\in S_r} \left(y_{s,n} * H_s\right) \geq L_r^{\text{Min}} \qquad \forall r \in \mathcal{R}^{W,\text{Hard}}, \forall n \in N_r$$

$$\sum_{s\in S_r} \left(y_{s,n} * H_s\right) \geq L_r^{\text{Min}} - h_{r,n}^{\text{Min}} \qquad \forall r \in \mathcal{R}^{W,\text{Soft}}, \forall n \in N_r$$

$$\sum_{s\in S_r} \left(y_{s,n} * H_s\right) \leq L_r^{\text{Max}} \qquad \forall r \in \mathcal{R}^{W,\text{Hard}}, \forall n \in N_r$$

$$\sum_{s\in S_r} \left(y_{s,n} * H_s\right) \leq L_r^{\text{Max}} + h_{r,n}^{\text{Max}} \qquad \forall r \in \mathcal{R}^{W,\text{Soft}}, \forall n \in N_r \tag{8}$$

Besides the hard constraints listed above, there are some constraints which can be formulated as either hard constraints or soft constraints (these are called *mixed constraints*). The soft constraints can be weighted by applying specific penalty costs to them in the objective function. The set of constraints (3) ensures that either the desired number of nurses are assigned to each activity or each violation is stored in the $c_{r,s}^{\text{Min}}$ and $c_{r,s}^{\text{Max}}$ variables respectively. The first two equations of the set of constraints (4) are required to *activate* and *deactivate* the group variables ($g_{r,g,n}^{\text{Group}}$) in order to correctly depict the activation state of a shift type group $g \in G_r$ of the rule r for the nurse $n \in N_r$. A shift type group (also called a *complete group*) is only activated if all given days of the shift type group are covered by shifts from that shift type group, for example we use these constraints to check whether a nurse works on the whole weekend. But if we only want to check whether a nurse works on at least 1 day of the weekend, then in this case a group (also called a *normal group*) is active if at least one shift from the group has been assigned. For these groups, the $D_g - 1$ expression from the first equation of (4) is not necessary and D_g from the second equation can be replaced by 1, leading to the following equations:

$$\sum_{s\in g} y_{s,n} \leq g_{r,g,n}^{\text{Group}} * M \qquad \forall r \in \mathcal{R}^G, \forall g \in G_r, \forall n \in N_r$$

$$\left(1 - g_{r,g,n}^{\text{Group}}\right) * M \geq 1 - \sum_{s\in g} y_{s,n} \qquad \forall r \in \mathcal{R}^G, \forall g \in G_r, \forall n \in N_r$$

Table 2 shows the activation and deactivation constraints working as intended for three arbitrary shifts ($s \in g$) on 3 different days ($D_g = 3$) for an arbitrary nurse n. As the shifts are interchangeable, only the sum of their y variables is important, not the individual values. Therefore, both the activation and deactivation constraints are required, as otherwise the group variable could be set to one for inactive groups (e.g., row 2 for normal groups) and vice versa (e.g., row 7 for complete groups). The remaining constraints of (4) simply use the group counter to enforce the lower and upper bounds, or to count the violations in the case of soft restrictions. Similarly, the set of constraints (5) guarantee the provision of correct coverage for a shift group.

Each undesired combination of activities stored in $S_{r,d}$ is either forbidden or related to a variable $v_{r,d,n}$ to count the violations in the set of constraints (6). The amount of assigned

Table 2 States of the shift group constraint

I	$g^{\text{Group}}_{r,g,n}$	Complete groups					Normal groups			
		II	I ≤ II	III	IV	III ≥ IV	V	I ≤ V	VI	III ≥ VI
0	0	2	✓	M	3	✓	0	✓	1	✓
0	1	M	✓	0	3	↯	M	✓	1	↯
1	0	2	✓	M	2	✓	0	↯	0	✓
1	1	M	✓	0	2	↯	M	✓	0	✓
2	0	2	✓	M	1	✓	0	↯	−1	✓
2	1	M	✓	0	1	↯	M	✓	−1	✓
3	0	2	↯	M	0	✓	0	↯	−2	✓
3	1	M	✓	0	0	✓	M	✓	−2	✓

I $\sum_{s\in g} y_{s,n}$; IV $D_g - \sum_{s\in g} y_{s,n}$
II $D_g - 1 + g^{\text{Group}}_{r,g,n} * M$; V, $g^{\text{Group}}_{r,g,n} * M$
III $\left(1 - g^{\text{Group}}_{r,g,n}\right) * M$; VI, $1 - \sum_{s\in g} y_{s,n}$

activities for each nurse is checked in the set of constraints (7), where it is ensured that the upper and lower bounds are not exceeded or all violations are stored in the $\omega^{\text{Min}}_{r,n}$ and $\omega^{\text{Max}}_{r,n}$ variables for the soft constraints. The set of working time constraints (8) consider not only the work-related activities, but also the others, which are defined to given working times. For example, a vacation activity usually has a working time of a fifth of the contractual weekly working time. Moreover, for soft restrictions the deviation from the lower and upper bounds is stored in the $h^{\text{Min}}_{r,n}$ and $h^{\text{Max}}_{r,n}$ variables to be considered in the objective function.

4.1.3 *Soft constraints*

$$\begin{aligned}
&\sum_{s\in\mathcal{S}_w} \left(y_{s,n} * H_s\right) \leq W_{w,n} * (1 + A_r) + e^{+}_{r,w,n} && \forall w \in \mathcal{W}, \forall r \in \mathcal{R}^H, \forall n \in N_r\\
&\sum_{s\in\mathcal{S}_w} \left(y_{s,n} * H_s\right) \geq W_{w,n} * (1 - A_r) - e^{-}_{w,n} && \forall w \in \mathcal{W}, \forall r \in \mathcal{R}^H, \forall n \in N_r
\end{aligned} \tag{9}$$

$$\begin{aligned}
&e^{+}_{r,w,n} \leq e^{\text{Week},+}_{r,w} && \forall w \in \mathcal{W}, \forall r \in \mathcal{R}^H, \forall n \in N_r\\
&e^{-}_{r,w,n} \leq e^{\text{Week},-}_{r,w} && \forall w \in \mathcal{W}, \forall r \in \mathcal{R}^H, \forall n \in N_r
\end{aligned} \tag{10}$$

The set of constraints (9) determines the weekly working hours for each nurse by summing up the working hours (H_s) of the assigned activities for each week. If the sum exceeds the regular working time $W_{w,n}$ plus the allowed deviation, the overtime variable ($e^{+}_{r,w,n}$) is activated to fulfill the inequation. For example, if the weekly working time of a nurse n is 38 h and the week is incomplete with 5 days in the optimization period, plus the allowed deviation is 10% and the nurse has worked 32 h, then the resulting overtime is as follows:

$$\begin{aligned}
\overbrace{32h\frac{5}{7}}^{\Sigma} &\leq \overbrace{\left(38h * \frac{5}{7}\right)}^{W_{w,n}} * \left(1 + \overbrace{0.1\frac{5}{7}}^{A_r}\right) + e^{+}_{r,w,n}\\
32h &\leq 29.86h + e^{+}_{r,w,n}\\
2.14h &\leq e^{+}_{r,w,n}
\end{aligned}$$

The variable $e_{r,w,n}$ is unbounded, but as it is penalized in the objective function, the solver assigns the lowest possible value (Suhl and Mellouli 2013, p. 105) which is in this case 2.14. The shortfall case is working similarly. The deviation is allowed because it is often impossible to exactly comply with the contractual working hours using activities with different working times. Nevertheless, the allowed deviation can be set to zero if the contractual working hours should be matched by the actual working hours as closely as possible. Moreover, the maximum overtime and shortfall of all nurses in their weekly working hours is determined in the set of constraints (10). This is important for the fairness of the rostering, since the solver does not differentiate between a single nurse having to work 10 overtime hours and 10 nurses each having to work an additional hour of overtime. By penalizing the maximum overtime and shortfall with high costs, the unfair solutions become less attractive and the work will be evenly balanced. The maximum overtime and shortfall are determined simply by specifying that the corresponding variables must be greater than or equal to all the individual overtimes and shortfalls of the nurses. For the same reason as stated before, the lowest possible values are assigned to the variables, which are equal to the biggest individual values.

Note that the mixed and soft constraints belong to the categories $\gamma : P$ and $\gamma : L$.

4.1.4 Objective function

The objective function (the category $\gamma : G$) primarily adds up the variables counting the violations of all soft constraints multiplied with the violation costs C_r^R (12–17). Additionally, the costs of the used arcs are added (11).

$$\min z = \sum_{a \in \mathcal{A}} \left(x_a * C_a^A \right) \tag{11}$$

$$+ \sum_{r \in \mathcal{R}^{C,\text{Soft}}} \sum_{s \in S_r} \left(\left(c_{r,s}^{\text{Min}} + c_{r,s}^{\text{Max}} \right) * C_r^R \right) \tag{12}$$

$$+ \sum_{r \in \mathcal{R}^{G,\text{Soft}}} \sum_{n \in N_r} \left(\left(g_{r,n}^{\text{Min}} + g_{r,n}^{\text{Max}} \right) * C_r^R \right) \tag{13}$$

$$+ \sum_{r \in \mathcal{R}^{GC,\text{Soft}}} \sum_{g \in G_r} \left(\left(c_{r,g}^{\text{Group,Min}} + c_{r,g}^{\text{Group,Max}} \right) * C_r^R \right) \tag{14}$$

$$+ \sum_{r \in \mathcal{R}^{S,\text{Soft}}} \sum_{d \in \left\{ 1,\cdots,|\mathcal{D}|-D_r^{\text{Sequence}} \right\}} \sum_{n \in N_r} \left(v_{r,d,n} * C_r^R \right) \tag{15}$$

$$+ \sum_{r \in \mathcal{R}^{T,\text{Soft}}} \sum_{n \in N_r} \left(\left(\omega_{r,n}^{\text{Min}} + \omega_{r,n}^{\text{Max}} \right) * C_r^R \right) \tag{16}$$

$$+ \sum_{r \in \mathcal{R}^{W,\text{Soft}}} \sum_{n \in N_r} \left(\left(h_{r,n}^{\text{Min}} + h_{r,n}^{\text{Max}} \right) * C_r^R \right) \tag{17}$$

$$+ \sum_{r \in \mathcal{R}^{H}} \sum_{w \in \mathcal{W}} \sum_{n \in N_r} \left(e_{r,w,n}^{+} * C_r^R + e_{r,w,n}^{-} * U_r * C_r^R \right) \tag{18}$$

$$+ \sum_{r \in \mathcal{R}^{H}} \sum_{w \in \mathcal{W}} \left(e_{r,w}^{\text{Week},+} * T_r * C_r^R + e_{r,w}^{\text{Week},-} * T_r * U_r * C_r^R \right) \tag{19}$$

4.2 Compuational results

Table 3 depicts the results of the exact approach. The MIP model is tested for a set of publicly available benchmark instances (see http://www.cs.nott.ac.uk/~tec/NRP/) (above the line of Table 3) and three test instances generated from Vivendi PEP (below the line). Vivendi PEP is a staff scheduling software, provided by Connext Communication GmbH. The first two instances are based on the sample data used in Vivendi PEP. The main difference lies in the coverage definition: the *KHStation6-April* instances uses time-based coverage, while the *WohnbereichGelb-April* instance uses shift-based coverage instead. This explains the significant discrepancy concerning the amount of activities (and therefore arcs) between these two instances. This supports the observation of Vanden Berghe (2002) and Bilgin et al. (2012) that problem instances with time-based coverage are more complex and harder to solve. The third instance (*KHIntensiv*) is based on real-world data from one of the Vivendi PEP users. Unfortunately, the coverage demands are not specified by the user, so they have to be derived from the actual schedules. This narrows the comparability between the handmade and automatically created schedules. The identified coverage demands are specified as shift-based coverage. All three instances are created by using the rules defined in Vivendi PEP and a previous period of 15 days is considered. The rules considered in Vivendi PEP include similar shifts, a complete weekend (Friday 10 pm to Monday 6 am), and weekly working hours for the workload balancing. Additionally, the rules to avoid isolated work days or isolated days off, and to prevent working blocks of more than five days are considered. The weight of each violation is set according to the definition in Vivendi PEP.

Most instances are solved on a machine running Microsoft Windows Server 2012 R2 with an AMD Opteron 4171 HE processor (8 cores, 2.10 GHz) and 14 GB memory. The instances which have been solved on a different machine are marked accordingly. The MIP solvers Gurobi 5.50 and CPLEX 12.5 are used. The amounts of variables and constraints represent the state of the MIP after it was reduced by CPLEX, so if Gurobi is used these numbers may be slightly different.

The majority of the benchmark instances were solved optimally, matching the optimal solutions stated on the instances' website. This indicates that the model is working correctly. Some instances were not solved optimally due to the following reasons. The *BCDT-Sep* instance was stopped after three days, as CPLEX was unable to improve the solution for about two days. CPLEX was unable to solve the *WHPP* instance optimally because the machine ran out of memory. The *ORTEC02* instance utilizes a lot of quadratic restrictions. As those are not supported by the MIP, CPLEX found a solution it regarded to be optimal by evaluating the restrictions linearly, which was not optimal if the restrictions were evaluated correctly. Although other instances use quadratic restrictions as well, those were solved correctly, as in the optimal solutions no quadratic restrictions were violated more than once. If only the benchmark instances with a significant runtime are taken into account ($\geq$1 min), CPLEX outperformed Gurobi in all cases and solved the instances about 30% faster than Gurobi on average. However, Gurobi was able to find significantly better solutions for two of the test instances.

Equally important as the quality of the solution is the amount of time required to find a good and usable solution, as the user is not willing to wait several hours or even days to get a schedule. Times of up to 10 (or at most 15) min have been identified to be acceptable (Bilgin et al. 2012; Valouxis et al. 2012; Santos et al. 2016), but obviously shorter runtimes ($\leq$5 min) are even better (Burke et al. 2013). Figure 2 depicts the costs of the best solution found in relation to the time.

Table 3 Test instances and results of the exact approach

Instance	Model						MIP			Time	
	Nurses	Days	Shift types	Activities	Arcs	Restrictions	Variables	Constraints	Solution	Gurobi	CPLEX
Azaiez	13	28	3	84	3159	122	5559	3778	0[a]	21.22 s	13.80 s
BCDT-Sep	20	30	5	150	7611	394	13,156	12,206	170	–	21 h 28 m[2]
BCV-3.46.2	46	26	4	104	18,400	306	28,925	16,551	894[a]	21 m 49 s	7 m 57 s
BCV-4.13.1	13	29	5	145	9100	418	14,254	7844	10[a]	38.89 s	35.61 s
GPost	8	28	3	84	1944	161	4700	3748	5[a]	4 m 58 s	3 m 11 s
Ikegami-2Shift-DATA1	28	30	4	96	7116	1629	15,890	12,154	0[a]	4 m 26 s	3 m 33 s
Ikegami-3Shift-DATA1	25	36	5	154	11,411	2065	24,269	16,651	2[a]	1 h 14 m	1 h 12 m[2]
Ikegami-3Shift-DATA1.1	25	36	5	154	11,411	2065	24,269	16,651	3[a]	2 h 7 m	1 h 55 m
Ikegami-3Shift-DATA1.2	25	36	5	154	11,411	2065	24,269	16,651	3[a]	–	4 h 53 m
LLR	27	7	4	28	2592	50	3255	1392	301[a]	0.36 s	0.50 s
Millar-2Shift-DATA1	8	14	3	42	936	67	1824	1304	0[a]	2.19 s	1.19 s
Millar-2Shift-DATA1.1	8	14	3	42	936	65	1776	1256	0[a]	2.56 s	1.27 s
Musa	11	14	2	28	572	160	488	404	175[a]	0.27 s	0.70 s
ORTEC01	16	31	5	155	12,000	379	22,371	13,379	270[a]	–	12 h 11 m[2]
ORTEC02	16	31	6	160	11,864	379	22,068	13,186	370	–	11 h 20 m[2]
Ozkarahan	14	7	3	21	756	88	798	446	0[a]	0.11 s	0.97 s
QMC-2	19	28	4	112	8208	703	12,164	6692	29[a]	1.92 s	1.42 s
SINTEF	24	21	6	126	17,280	279	22,014	7974	0[a]	58.67 s	47.45 s
Valouxis-1	16	28	4	112	6912	182	11,480	6680	20[a]	4 h 49 m	4 h 03 m[2]
WHPP	30	14	4	56	6240	97	9894	5694	2000	–	1 h 20 m[1,2]

Table 3 continued

Instance	Model						MIP			Time	
	Nurses	Days	Shift types	Activities	Arcs	Restrictions	Variables	Constraints	Solution	Gurobi	CPLEX
KHStation6-April	12	45	49	462	56,904	2468	63,911	12,047	33,363.36	–	2 d 09 h[2]
									34,387.11	1 d 09 h	–
WohnbereichGelb-April	16	45	44	210	11,904	1136	16,069	7339	72,945.00	–	1 d 19 h
									61,835.00	1 d 13 h	–
KHIntensiv	44	4	69	352	20,585	3375	27,776	14,182	30,067.64	–	20 h 54 m
									25,886.39	2 d 13 h	–

[a] Optimal 1 out of memory 2 Intel Xeon E5-2660 (8 cores, 2.20GHz)

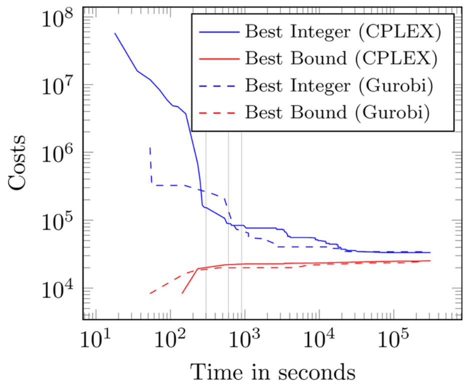

Fig. 2 Solution path of the *KHStation6-April* instance

Within the first 5 min (first vertical line) CPLEX was able to improve the quality of the solution extremely quickly, dropping the penalty costs from approximately 60 million to about 150,000. Until the 10- and 15-min mark (second and third vertical lines) the solution is improved to ~88,000 and ~77,000 respectively. After the 15-min mark it took over an hour to get to a significantly improved solution. The best solution (33,363.36) was found after two days, 9 h and 43 min. CPLEX was unable to improve this solution until it was terminated after about 3 1/2 days. At this point the optimization was stopped, as the best bound was increased only marginally over the whole duration, so it would probably have taken a lot of additional time to check whether the solution is optimal. The gap between the best MIP solution and the best bound at this point was 24.22%.

Gurobi starts with a better solution than CPLEX but is outperformed between the 5- and 10-min mark. After the 10-min mark it performs better than CPLEX and is only outperformed at the very end of the search. However, the solutions and bounds found by Gurobi and CPLEX are only slightly different.

Although the solutions found by CPLEX and Gurobi are not proven to be optimal, they can be used as reference to evaluate the performance of alternative solution approaches.

5 Simulated annealing

Simulated annealing is chosen as the alternative solution approach due to the following reasons. First, simulated annealing is fairly easy to implement (Gendreau and Potvin 2010, p. 1). This is helpful for the iterative approach, since the constraints are implemented and tested one by one. Additionally, it can be used in hybrid methods, if the simulated annealing alone does not yield acceptable results (e.g., as done by Bai et al. 2010 and Hadwan and Ayob 2010). There are other metaheuristics sharing these features (such as tabu search), but simulated annealing has been chosen since it has proven itself to be robust (Dowsland 1998), in the sense that its performance is not negatively influenced peculiarly by differences between the problem instances. This quality is important, as hospitals may change priorities and regulations (Ernst et al. 2004).

In the simulation annealing, a new solution is computed in each iteration by altering the current solution. Hence, a feasible initial solution must be computed before the simulated annealing algorithm starts. How this initial solution is obtained, which neighborhood operators and parameters will be used, and the results of this solution approach, are presented in the following subsections.

5.1 Initial solution

The process to find an initial solution is described as follows. At the beginning of the algorithm, a path is randomly searched in the network for each nurse (nurse by nurse), while all hard constraints are considered during the search. If we do not find such paths for all nurses, the algorithm tries all available *initial solution services*, which use specialized methods to find a feasible solution. Currently, the only available initial solution service uses a MIP solver and will be introduced later. Every service has at most 30 s to find a feasible solution, after which it is aborted and the next service is used. If any one succeeds, the solution of that service is used as the initial solution. If all services fail (or none is available), the algorithm converts (relaxes) the violated constraints to soft constraints with high costs. Naturally, these costs should be significantly higher than the maximum costs of all soft constraints. After that, the algorithm searches for a feasible path through the network for all nurses again.

The only initial solution service which has been implemented so far utilizes a simplified MIP model, which is solved by Cbc (Coin-or branch and cut, https://projects.coin-or.org/Cbc) to get a feasible solution. Cbc was chosen for the following reasons. First, it is one of two free MIP solvers which can be used commercially. Second, the other solver is Ip_solve (http://www.ateji.com/optimj/lpsolve), which needs significantly more effort to use, since it does not offer a .NET API. Additionally, Cbc is nearly twice as fast as Ip_solve (Zuse Institute Berlin 2014).

We use a MIP solver to find a feasible solution, since most of the constraints of our problem can be either soft or hard constraints. Some publications (e.g., Burke et al. 2001; Lü and Hao 2012 and Burke et al. 2013) have used lightweight methods to create an initial solution, because in those cases the soft and hard constraints were strictly defined. Therefore, the characteristics of the hard constraints could be taken into account while searching for a feasible solution (e.g., in Lü and Hao 2012; Burke et al. 2013, all coverage constraints are hard, so a feasible schedule can also be achieved by simply assigning enough nurses to the shifts). However, in our case utilizing the characteristics of the constraints would require significant effort. Valouxis et al. (2012) use an Integer Programming (IP) model to find an initial solution as well. However, they did not try to find a solution for the original problem but only assigned "work day" to each nurse. Concrete shifts were assigned on each of the nurses' work days in a second step. That approach can only be used reasonably if all shift types share an equal (or at least similar) working time. However, we cannot use their approach for our problem, since the prerequisite of all shift types having an equal working time is not given.

5.2 Operators

This subsection introduces the different operators listed as follows, which are applied to small to medium-sized neighborhoods.

- *TwoWaySwap* Swaps the activities of two random nurses on a random day. If this operation is chosen, it is decided randomly whether it is performed once or twice in that iteration.
- *ThreeWaySwap* Swaps the activities of three random nurses on a random day.
- *SwapSequences* Swaps the sequence of activities of a random length (2–6) starting on a random day between two random nurses.
- *RandomSwap* Assigns a different activity to a nurse on 1 or 2 random days.
- *DayOff* Assigns the day off activity to a random nurse on 1 or 2 random days.

The *TwoWaySwap* and *SwapSequences* operators are mostly used in the literature to solve the scheduling problem and they are proven to provide good results (Qu and He 2010;

Burke et al. 2013), although the *TwoWaySwap* operator should not be used solely since it may perform poorly to remove violations of certain kinds of constraints (auf'm Hofe 2001; Burke et al. 2013). However, we implement it as well, since it is a relatively simple and fast operation and has the best success rate for our problem. An operator is considered to be successful in case its produced solution is better than the current solution. Additionally, we use the *RandomSwap* operator to avoid the flaw of the previous two operators, which only changes activities between nurses, but do not change the numbers of them. For example, if an early shift on one day is overcovered in the initial solution, then the *RandomSwap* is required to correct the coverage of that shift. There is a possibility that this shift is replaced by a day off activity. However, the probability that the day off activity is chosen as replacement is relatively low. This issue can be solved by the *DayOff* operator, which works like the *RandomSwap* operator but always assigns the day off activity. The *ThreeWaySwap* operator is also implemented in our work because it works well for Stølevik et al. (2011).

In each iteration one of those operators is chosen randomly to generate a new solution. The probability of using each of the operators was chosen according to their success rate. On average, the *TwoWaySwap* operator had the highest success rate (∼20%), so it was chosen with a probability of 30%. It was followed by the *SwapSequences* operator, each sequence length of that was allocated with a probability of 8% (in total 40% combined for all sequence lengths). Note that, we calculate the success rate of the *SwapSequences* operator with the average success rate of all five sequence lengths. The *ThreeWaySwap* operator had the lowest success rate (∼15%), so the probability was set to 10%. The remaining 20% were split 15% to 5% between the *RandomSwap* and *DayOff* operators, because the latter is only a specialized version of the former.

5.3 Parameter tuning

Apart from the neighborhood function, the two main factors which influence the performance of simulated annealing are the initial temperature and the cooling schedule (Michalewicz and Fogel 2004, p. 121). Using a fixed value for the initial temperature is not advisable since it should correlate with the values used to weight the violation of soft constraints, so that the simulated annealing can work as intended (i.e., a high probability of accepting inferior solutions during the hot phase at the beginning). For example, using an initial temperature of 1000 can work if the highest weighted constraint has a value of 100, but if it has a value of 10,000 the probability of accepting a solution violating that constraint is very low ($<1\%$) at all phases of the simulated annealing, limiting its ability to escape local optima. Hence, the initial temperature must be determined dynamically from the problem instance. This is achieved by calculating the maximum possible degradation of the objective value which can occur by swapping a single activity, i.e., the value of the maximum of the sums of the weights from the constraints attached to each nurse-activity combination; and the maximum arc costs are calculated with a multiplier to get the initial temperature. A simple cooling schedule is used, which decreases the temperature by multiplying the current temperature by a certain factor (≤ 1) after 30,000 iterations (note that, the number of iterations is chosen based on our experiments, so the temperature of the large instances can also be reduced within a second).

To find the dominant multiplier and cooling factor, several combinations were tested to see which combination yields the best results on average in an acceptable amount of time. The values for the multiplier are tested between 5 and 30, while the values for the cooling factor between 0.5 and 0.9 are tested. Higher values were not tested since they led to long runtimes due to a very long hot (random) phase. Lower values were omitted since they led to low probabilities of accepting inferior solutions, causing the simulated annealing to act more

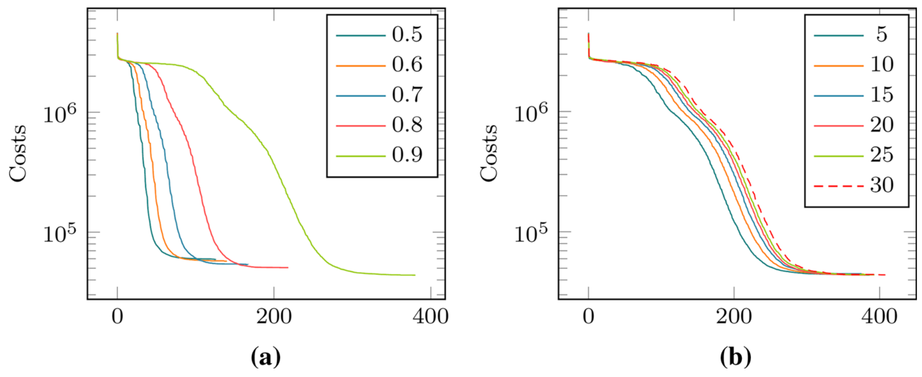

Fig. 3 Average solution values for the *KHStation6-April* instance. **a** A fixed multiplier of 20. **b** A fixed cooling factor of 0.9

like a hill climber. All possible multiplier and cooling factor combinations are tested on the *GPost*, *KHStation6-April*, *WohnbereichGelb-April* and *KHIntensiv* instances. To get viable results, the simulated annealing was run 30 times for each combination. It became apparent that the cooling factor is the deciding factor for both the runtime and the average quality of the solution. Figure 3a depicts the solution values for the *KHStation6-April* instance with a fixed multiplier of 20 and different cooling factors. It shows that the average solutions values and especially the runtimes vary significantly. Unfortunately, the cooling factor leading to the best solutions on average (0.9) also causes the longest runtimes by far. Figure 3b depicts the opposite case with a fixed cooling factor and different multipliers. It shows that the runtimes and the quality of the average solutions only depends slightly on the multiplier. Consequently, the best overall solutions for all test instances were found using a cooling factor of 0.9. The best multiplier according to the results is 20, since it led to the best results on average for three of the four instances.

5.4 Results

The performance of the simulated annealing is evaluated using the multiplier and cooling factor determined in the previous subsection. All instances are solved 50 times on a slightly faster machine[1] than the one used for the exact approach. However, the simulated annealing only uses one of the CPU's cores, while the MIP solvers utilize all eight cores. Therefore, the computational times consumed by our simulated annealing and the MIP solvers cannot be used directly for the purpose of comparison. The search of the simulated annealing is terminated if no new solution is found for 500,000 iterations and the temperature drops below 10. The second condition is necessary, since the search sometimes stops preemptively during the hot phase in case no new better solution is found for a longer period of time. Increasing the number of iterations without a new solution to circumvent this issue would have been ineffective, since it would have prolonged the search at the end unnecessarily. Alternatively, the search is terminated if a solution with an objective value of zero is found.

Table 4 shows the best, average and worst results together with the average solving time for all instances. In addition, it contains the results published by other researchers.

All small instances (those instances which were solved in under a minute by the MIP solvers) are solved optimally by the simulated annealing. Three instances (*Millar-2Shift-*

[1] Microsoft Windows Server 2012 R2; Intel Xeon CPU E5-2660 (8 cores, 2.20 GHz); 14 GB memory.

Table 4 Results of simulated annealing

Instance	Simulated annealing				SSM[1]		VNS|ALNS[2]	SVNA[3]				BP[4]	
	Best	Average	Worst	Avg. time	Best	Time	Average	Best	Average	Worst	Avg. time	Best	Time
Azaiez	0[a]	2.52	8	4 m 25 s	–	–	–	–	–	–	–	0[a]	0.3 s
BCDT-Sep	220	297.00	350	5 m 18 s	–	–	–	–	–	–	–	100[a]	1 h 43 m
BCV-3.46.2	894[a]	894.76	897	6 m 19 s	894	1 h	930.60	894[a]	894[a]	894[a]	25 m	894[a]	8.3 s
BCV-4.13.1	10[a]	10.10	12	6 m 43 s	10[a]	29 m 35 s	52.20	10[a]	10[a]	10[a]	7.1 s	10[a]	<30 s
GPost	20	555.96	1616	4 m 30 s	9	1 h 11 m	–	–	–	–	–	5[a]	2 s
Ikegami-2Shift-DATA1	0[a]	1.18	4	6 m 18 s	–	–	–	–	–	–	–	0[a]	41.7 s
Ikegami-3Shift-DATA1	58	89.76	194	8 m 20 s	–	–	–	–	–	–	–	2[a]	9 m 58 s
Ikegami-3Shift-DATA1.1	67	97.48	235	8 m 06 s	–	–	–	–	–	–	–	4	16 m 35 s
Ikegami-3Shift-DATA1.2	59	94.62	185	8 m 18 s	–	–	–	–	–	–	–	5	1 h 30 m
LLR	301[a]	301.36	302	1 m 44 s	301[a]	35 m 15 s	–	301[a]	301.48	305	1.85 s	301[a]	0.8 s
Millar-2Shift-DATA1	0[a]	94.00	400	1 m 36 s	0	15 m 10 s	–	–	–	–	–	0[a]	<0.1 s
Millar-2Shift-DATA1.1	0[a]	0.00[a]	0[a]	1 m 22 s	0	20 s	–	–	–	–	–	0[a]	<0.1 s
Musa	175[a]	175.00[a]	175[a]	1 m 32 s	–	–	–	175[a]	175[a]	175[a]	0.22 s	175[a]	<0.1 s
ORTEC01	305	731.28	1360	8 m 45 s	365	56 m 40 s	–	–	–	–	–	270[a]	1 m 9 s
ORTEC02	351	901.84	2476	8 m 26 s	–	–	–	–	–	–	–	270[a]	1 m 45 s
Ozkarahan	0[a]	0.00[a]	0[a]	32 s	–	–	–	–	–	–	–	0[a]	<0.1 s
QMC-2	29[a]	56.90	259	3 m 42 s	–	–	–	–	–	–	–	29[a]	1.9 s
SINTEF	0[a]	11.06	89	3 m 25 s	4	1 h 08 m	16.40	–	–	–	–	0[a]	10.5 s
Valouxis-1	40	118.80	180	4 m 22 s	100	1 h 06 m	1604.00	20[a]	73.33	120	90.19 s	80	15 m 10 s
WHPP	3000	3474.64	4120	4 m 52 s	–	–	–	5[a]	5[a]	5[a]	16.2 s	5[a]	17.6 s
KHStation6-April	37,414.43	44,397.77	51,309.21	7 m 28 s	–	–	–	–	–	–	–	–	–
WohnbereichGelb-April	69,235	76,994.01	82,348.57	6 m 54 s	–	–	–	–	–	–	–	–	–
KHIntensiv	35,048.28	40,894.11	47,152.28	8 m 20 s	–	–	–	–	–	–	–	–	–

[a] Optimal; 1, Burke et al. (2009); 2, Bilgin et al. (2012); 3, Solos et al. (2013); 4, Burke and Curtois (2014)

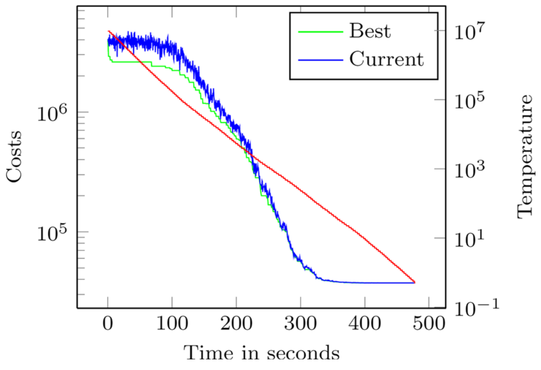

Fig. 4 Solution value progression of the *KHStation6-April* instance

DATA1.1, *Musa* and *Ozkarahan*) are even solved optimally every time, which is remarkable because of the randomness of the algorithm. The results for the *Millar-2Shift-Data1* instance are interesting, since it is either solved optimally (0) or relatively poorly (400). This indicates that the simulated annealing has issues to escape the local optimum leading to a bad solution. It is possible that the global and local optimums found themselves on opposite edges of the solution space, so that is decided relatively early in the solving process which optimum will be reached. For most of the larger instances the simulated annealing performs fairly well, finding good solutions on average. Although good solutions are found for the *GPost*, *ORTEC01* and *ORTEC02* instances, the average performance for those instances is not satisfactory. These three instances are using strict coverage definitions (i.e., the exact coverage demand is specified for each shift, and any kind of over- and undercoverages are penalized harshly), so this could indicate that the simulated annealing works better if the coverage definitions allow some flexibilities. There is no obvious reason for the poor performance in solving the *WHPP* instance. It also uses strict coverage definitions, but it is a relatively small instance. Therefore, the difficulties that simulated annealing (and CPLEX) had in trying to solve this instance cannot really be explained.

The performance of the simulated annealing for the test instances (below the line) is surprisingly good. For one instance (*WohnbereichGelb-April*) it even finds a better solution than CPLEX. None of the solutions violate any soft constraints, which are highly weighted, such as legally required resting times and maximum weekly working times, and they always provide at least the minimum required coverage. Additionally, the approach used to balance the workload works quite well, effectively reducing the amount of over- and undertime.

Figure 4 depicts the solution value progression and temperature over time for the best solution found for the *KHStation6-April* instance. The different phases of the simulated annealing can be easily spotted.

The amount of time required to find a solution is roughly the same for all instances. This can be explained because one of the stopping criteria requires the temperature to be below 10. Therefore, the runtime is determined by the number of constraints and the amount of iterations which can be computed each second. The instances with significantly shorter runtimes are those where the search was ended preemptively (because a solution with an objective value of zero was found, i.e., the optimal solution was found), or the very small instances (*LLR* and *Musa*).

Compared with the published results, the simulated annealing performs fairly well. It found equal or better solutions for all except the *GPost* instance, in comparison with the scatter search methodology (SSM) by Burke et al. (2009). For all except the *Millar-2Shift-DATA1.1* instance, the solutions are also found after a significantly shorter amount of time. Bilgin et al. (2012) do not mention the exact times in their results. They only state that their aim was to keep the runtimes below 15 min. The simulated annealing was able to find better solutions on average for all except the *WHPP* instance in comparison to their variable neighborhood search (VNS) and adaptive length neighborhood search (ALNS). However, these results may be disleading since they interprete the benchmark instances slightly differently (e.g., they add some restrictions at the start of the planning period). The stochastic variable neighborhood approach by Solos et al. (2013) outperforms the simulated annealing in all cases except for the *BCV-3.46.2* instance. Unfortunately, they did not test one of the large instances (*Ikegami* or *ORTEC*). It would have been interesting to see how their algorithm handles those instances. The Branch and Price method used by Burke and Curtois (2014) apparently works very well for the benchmark instances, as it finds the optimal solution for all instances after relatively short times. Only the *Valouxis-1* instance is solved better by the simulated annealing. The authors do not find an obvious reason why Branch and Price performed poorly for this instance (Burke and Curtois 2014). However, if the available runtime is limited, the simulated annealing outperforms the Branch and Price in some cases (e.g., the best solution found for the *BCDT-Sep* instance after 10 min has an objective value of 330, and for a time limit of 30 s it is 500). Hence, the simulated annealing may outperform the Branch and Price method if the runtime is limited, especially considering that the simulated annealing is able to solve all of the instances in less than 2 min on a modern computer. Whether this is the case cannot be answered conclusively, since Burke and Curtois (2014) only publish their results for a runtime of 30 s and 10 min, so the intermediate results are not available. Additionally, they performe their tests on a slower CPU.

6 Summary and outlook

We introduce in this work a flexible problem model for the Nurse Rostering Problem, and to the best of our knowledge, it is the first one to be classified as ASBN|RVNTO|PLG according to the $\alpha|\beta|\gamma$ notation by Causmaecker and Vanden Berghe (2010). It is flexible enough to model most of the available benchmark instances and the majority of the regulations which are defined in Vivendi PEP. In contrast to most of the problem models proposed in the literature, it provides the possibility to use the nurses' contractual working hours to balance the workload, which reduces the configuration effort for the user, and also allows modeling of temporary employees. Two solution approaches are presented to solve problem instances.

The first one is an exact approach using a MIP model and solvers. With this approach we can solve most of the benchmarking instances optimally, proving that the mathematical model accurately describes the benchmarking instances and finds good solutions for the test instances created from Vivendi PEP.

The second one is a heuristical approach using simulated annealing. It shows that it produces good results for both the benchmarking and the test instances in a short amount of time, and it also proves that it is robust concerning different problem instances. In a time-restricted environment, it is able to compete with the majority of the solution approaches presented in the literature. However, it does not adapt well to a fixed given running time. On one hand, if the user wants to wait longer to get a better solution, it does not use the additional time efficiently to further enhance the quality of the solution. On the other Hand, it does not adapt to situations where a solution must be produced in a very short amount of time. We have tried to solve the first problem by applying two additional solution methods to the solution

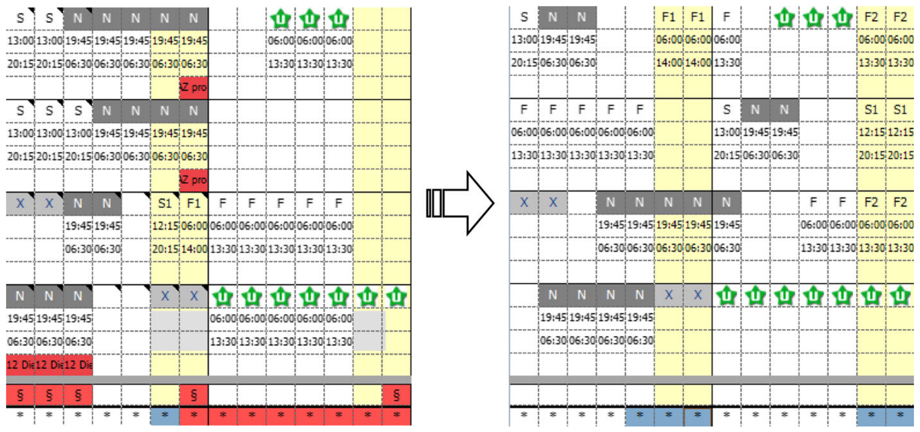

Fig. 5 Manually (*left*) and automatically created schedules for the *KHIntensiv* instance

created by the simulated annealing. Unfortunately, both methods, the first using constraint programming and the second a variable depth search (see "Appendix" section), are not able to significantly improve the solution. The second issue can be resolved by changing the parameters (multiplier and cooling factor) according to the available time. It was shown that decreasing the cooling factor leads to significantly shorter runtimes. However, this decreases the quality of the solution in most cases.

As was expected, using the exact approach is not an option for a practical application, not only because of the license costs for the MIP solvers, but also because of the amount of time required to find a good (and not optimal) solution. Gurobi requires more than 30 min to find a solution matching the worst solution found by simulated annealing for the *KHStation6-April* instance quality-wise. CPLEX needs more than 2 1/2 h for the same result. To find a better solution than simulated annealing, both the CPLEX and Gurobi solvers need more than 5 h.

The solution methods implemented in this work are integrated into Vivendi PEP, giving the user the possibility to create the problem model including the regulations, fixed assignments and individual wishes (e.g., desired shifts) directly from Vivendi PEP's planning view. A rudimentary GUI was implemented to choose the solution method, set a time limit and adjust the weights of the constraints. The automatically created schedule is presented in the planning view, so that it can be reviewed and adjusted when necessary.

Figure 5 shows an excerpt from Vivendi PEP's planning view, containing the schedule for the real-world instance *KHIntensiv*. On the left-hand side, the manually created schedule is shown. Several constraints are violated (marked with the color red), including an exceedance of the allowed weekly working hours and undercoverage. The figure on the right-hand side shows the schedule that is created by simulated annealing. None of the constraints are violated (not only in the excerpt but in the whole schedule). Furthermore, all coverage demands are fulfilled and none of the fixed assignments [vacation (U) and forced day off (X)] are changed. The color blue in the figure indicates the overstaffing; as mentioned in Sect. 2, the case of the overlapping (e.g., 15 min) at the time of the team switch is desired for a short briefing. According to Burke and Curtois (2014), all the solutions found by their variable depth search heuristic are far superior to all manually created schedules. Their statement is backed by our finding that the quality of the solutions found by the simulating annealing is better than that of the manually created ones.

Future work should concentrate on increasing the usability of the solution, e.g., by providing an easy way to add restrictions which cannot be modeled in Vivendi PEP, and to adjust the weights of the restrictions. Furthermore, the flexibility of the problem model could be enhanced, e.g., to allow assigning several shifts per day to a single nurse, which is allowed in Vivendi PEP. Furthermore, it is possible to implement and evaluate additional metaheuristics with relatively low effort, since the implementation of the constraints can be reused.

Acknowledgements We would like to thank Connext Communication GmbH for providing us with the real-world instances and the use of their software, Vivendi PEP, to accomplish this paper.

Appendix: Alternative solution apporaches

The simulated annealing (SA) approach is proven to be effective in finding good solutions in a short amount of time. However, it is not able to utilize additional available time, if the user is willing to wait longer to get a better solution. We have tried several methods to deal with this situation. First, the temperature is increased again at the end of the search, with the purpose to escape the local optimum to get to a improved solution. However, this is proven to be ineffective, since the quality of the solutions is only improved marginally or not at all. Destroying the solution before reheating (i.e., removing the assignments in a randomly selected part of the planning period for a subset of nurses) has no improving effect either. An explanation for the poor performance of those methods might be given by Burke et al. (2013). They observe that it is necessary to chain several neighborhood moves to escape a local optimum. Due to the randomness of SA, it is unlikely that the correct moves are chained together. Hence, alternative approaches are evaluated, to be used in cooperation with or as replacement for SA in the following subsection, namely constraint programming and variable depth search.

Constraint programming

Constraint programming (CP) is mostly used for constraint satisfaction problems (CSP), which focus on finding a feasible solution to a problem and not on optimizing an objective value (van Omme et al. 2013, p. 9). Consequently, we use CP firstly to generate an additional initial solution service. Modeling a problem as CSP is similar to modeling it as MIP. In both cases, variables and constraints are used to describe the problem. However, in some cases modeling a CSP may be easier, since so-called *global constraints* can be used (van Omme et al. 2013, p. 10). For example, one of these constraints, the *AllDifferent* constraint, can be used to define that a set of variables must all have different values. To model such a constraint in a MIP model requires a significantly higher effort. Additionally, the global constraints can be checked very efficiently (van Omme et al. 2013, p. 10). The CSP is implemented and solved using the CP solver. The first results look promising. The CP solver is able to find a feasible solution for a problem instance using hard constraints solely for coverage definitions more or less instantly (<0.1 s). However, as soon as more hard constraints were added, the CP solver was not able to find a feasible solution even after an extended amount of running time (>30 min). This is because the CP-solver uses a method called *branch and prune*, where it traverses a search tree. In the search tree, the search space is divided at each branch and all subtrees which cannot contain a feasible solution are pruned (van Omme et al. 2013, pp. 10, 84, 94). The solver backtracks if a subtree has not led to a feasible solution, and starts to explore another subtree (van Omme et al. 2013, p. 11). The backtracking occurs frequently due to the combinations of the hard constraints, e.g., the maximum and minimum work-related activities per week and the maximum work-related activities for the whole planning period; moreover, the search space for the real-world instances is huge (for example *GPost* has a search space with a size of 10^{107}). Therefore, a huge amount of time is required for backtracking. We have tried to change the order of the variables and search strategy to reduce the amount of backtracking; however, the outcome does not change significantly.

Stølevik et al. (2011) use constraint programming for a similar purpose. They experience similar issues with some combinations of hard constraints. In those cases they created a "basic solution" instead, which only satisfies a subset of the hard constraints. They were able to prioritize certain hard constraints, because they defined the set of hard and soft constraints explicitly. Defining mixed constraints is not possible using their problem model. Additionally, they use a more sophisticated variable ordering and value selection method to exploit information, which is not possible for the problem instances used in this paper (e.g., a desired number of shift types to be assigned to each nurse).

Qu and He (2010) utilize constraint programming to create a good sequence of activities which do not violate any of the combination constraints. However, they only consider problem instances with at most four shift types. Their approach is not applicable to problem instances with significantly more shift types, since the number of possible combinations grows exponentially.

Additionally, we also try constraint programming to enhance the quality of the solution found by SA. For this purpose the problem is modeled as s constraint optimization problem (COP) and solved by the CP-solver. The idea is to use the objective value of the best solution found by SA to reduce the search space, while pruning all subtrees leading to a worse solution. However, this does not have a significant influence on the quality.

Variable depth search

The second approach chosen for improving the solutions achieved by SA is a variable depth search (VDS) algorithm presented by Burke et al. (2013), since it uses compound neighborhood moves to escape local optima. Two neighborhoods are used by the authors. First, a small neighborhood which consists of swapping all activity sequences of length two or three between all nurses. This neighborhood is explored until a new best solution is found, or a solution which increases the quality of a nurse's individual schedule (according to the horizontal rules). If a new best solution is found, the search restarts outgoing from this new solution. Otherwise a bigger neighborhood is explored which consists of swapping all activity sequences of length five or six between the nurse, which suffers a degradation to her individual schedule, and all other nurses. After the whole neighborhood has been explored, the move which leads to a new best solution (in that case the search restarts) or the move which increases the quality of the nurse's schedule the most is performed. If no such move exists the search continues using the small neighborhood. Otherwise, the search continues using the large neighborhood for the nurse whose schedule was worsened by the last swap. These steps are repeated until a maximum predefined depth (maximum chain length) has been reached. If no new best solution has been found at this point, all steps are undone and the search continues in the small neighborhood, restarting from the best solution.

The algorithm is implemented using the *worsened day heuristic*, which restricts the search in the large neighborhood to swaps which include a day which was *worsened* by a previous swap. Our implementation could only replicate the results observed by Burke et al. (2013) to some extent. For the *ORCTEC01* instance, an average objective value of 1604 (maximum depth: 1000) is found, compared with the 1120 of Burke et al. (2013). This discrepancy is caused by the unclear *worsened* situations, such as, an activity is assigned which violates a forbidden sequence but it is unclear which days should be marked as *worsened*: all days of the shift sequence or only the day of the assigned shift.

Additionally, due to the nature of the neighborhoods used, the VDS can only be applied effectively for problem instances with strict coverage definitions. Therefore, using it with randomly created initial solutions is not an option. Applying the VDS to problem instances with variable coverage definitions can only work if a good (or optimal) amount of nurses are assigned to all activities in the initial solution. We do not try another neighborhood which might resolve this issue, since the overall performance of the VDS, especially for medium and large instances, was surpassed by the SA algorithm. For example, the best and average objective values for the *ORTEC01* instance found by the VDS are 480 and 1120, compared the 305 and 731.28 found by SA.

Although the VDS does not perform well on its own, we try to use it to improve the solution found by SA, since the solution is assumed to already have a good degree of coverage. However, this has not proved to be time-worthy, since the VDS is only able to improve the solution marginally.

References

Aickelin, U., & Dowsland, K. (2008). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *3*(3):139–153 (arXiv preprint arXiv:0802.2001).

auf'm Hofe, H. M. (2001). Solving rostering tasks by generic methods for constraint optimization. *International Journal of Foundations of Computer Science*, *12*(05), 671–693.

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.

BDI. (2013). *Die Gesundheitswirtschaft ein stabiler Wachstumsfaktor für Deutschlands Zukunft*. http://bit.ly/1m4qf0M

Bilgin, B., De Causmaecker, P., Rossie, B., & Vanden Berghe, G. (2012). Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, *194*(1), 33–57.

Burke, E., De Causmaecker, P., & Vanden Berghe, G. (1998). A hybrid tabu search algorithm for the nurse rostering problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, Springer (pp. 187–194).

Burke, E. K., De Causmaecker, P., & Vanden Berghe, G. (2004a) Novel meta-heuristic approaches to nurse rostering problems in belgian hospitals Problems in Belgian Hospitals. In J. Leung (Ed.) *Handbook of scheduling: algorithms, models and performance analysis*. Citeseer

Burke, E. K., Causmaecker, P. D., Petrovic, S., & Vanden Berghe, G. (2006). Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence*, *20*(9), 743–766.

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

Burke, E., Cowling, P., De Causmaecker, P., & Vanden Berghe, G. (2001). A memetic approach to the nurse rostering problem. *Applied Intelligence*, *15*(3), 199–214.

Burke, E. K., & Curtois, T. (2014). New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, *237*(1), 71–81.

Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B. (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, *188*(2), 330–341.

Burke, E. K., Curtois, T., Qu, R., & Vanden Berghe, G. (2009). A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, *61*(11), 1667–1679.

Burke, E. K., Curtois, T., Qu, R., & Vanden Berghe, G. (2013). A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, *25*(3), 411–419.

Burke, E. K., Li, J., & Qu, R. (2010). A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, *203*(2), 484–493.

Cappanera, P., & Gallo, G. (2004). A multicommodity flow approach to the crew rostering problem. *Operations Research*, *52*(4), 583–596.

Causmaecker, P., & Vanden Berghe, G. (2010). A categorisation of nurse rostering problems. *Journal of Scheduling*, *14*(1), 3–16.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems-a bibliographic survey. *European Journal of Operational Research*, *151*(3), 447–460.

Dowsland, K. A. (1998). Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, *106*(2–3), 393–407.

Drake, R. G. (2014). The nurse rostering problem: From operational research to organizational reality? *Journal of Advanced Nursing*, *70*(4), 800–810.

Ernst, A., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, *153*(1), 3–27.

Gendreau, M., & Potvin, J. Y. (2010). *Handbook of metaheuristics. International series in operations research and management science* (Vol. 146). New York: Springer.

Hadwan, M., & Ayob, M. (2010). A constructive shift patterns approach with simulated annealing for nurse rostering problem. In *Information Technology ITSim 2010 International Symposium in 1*.

Haspeslagh, S., De Causmaecker, P., Schaerf, A., & Stølevik, M. (2012). The first international nurse rostering competition 2010. *Annals of Operations Research*, *218*, 221–236.

He, F., & Qu, R. (2012). A constraint programming based column generation approach to nurse rostering problems. *Computers & Operations Research*, *39*(12), 3331–3343.

Kellogg, D. L., & Walczak, S. (2007). Nurse scheduling: From academia to implementation or not? *Interfaces*, *37*(4), 355–369.

Lim, G. J., Mobasher, A., Kardar, L., & Cote, M. J. (2012). *Handbook of healthcare system scheduling. International series in operations research and management science* (Vol. 168). New York: Springer.

Lü, Z., & Hao, J. K. (2012). Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, *218*(3), 865–876.

Maenhout, B., & Vanhoucke, M. (2009). Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, *13*(1), 77–93.

Michalewicz, Z., & Fogel, D. B. (2004). *How to solve it: Modern heuristics*. Berlin: Springer.

Online Z. (2013). *Fachkräftemangel - regierung wirbt um ausländische pflegekräfte*. http://bit.ly/1oUIDhj

Osogami, T., & Imai, H. (2000). Classification of various neighborhood operations for the nurse scheduling problem. *Lecture Notes in Computer Science*, *1969*, 72–83.

Qu, R., & He, F. (2010). A hybrid constraint programming approach for nurse rostering problems. *European Journal of Operational Research*, *203*(2), 211–224.

Santos, H. G., Toffolo, T. A., Gomes, R. A., & Ribas, S. (2016). Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, *239*(1), 225–251.

Smet, P., Brucker, P., De Causmaecker, P., & Vanden Berghe, G. (2014). Polynomially solvable formulations for a class of nurse rostering problems. In *Proceedings of the 10th international conference on the practice and theory of automated timetabling* (pp. 408–419).

Solos, I., Tassopoulos, I., & Beligiannis, G. (2013). A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem. *Algorithms*, *6*(2), 278–308.

Stølevik, M., Nordlander, T. E., Riise, A., Frøyseth, H. (2011). A hybrid approach for solving real-world nurse rostering problems. In *International Conference on Principles and Practice of Constraint Programming*, Springer (pp. 85–99).

Suhl, L., & Mellouli, T. (2013). *Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen*. Berlin: Springer.

Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., & Housos, E. (2012). A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, *219*(2), 425–433.

van Omme, N., Perron, L., & Furnon, V. (2013). *Or-tools users manual*. Technical reports, Google

Vanden Berghe, G. (2002). An advanced model and novel meta-heuristic solution methods to personnel scheduling in healthcare. https://lirias.kuleuven.be/handle/123456789/249444.

Wright, P. D., Bretthauer, K. M., & Côté, M. J. (2006). Reexamining the nurse scheduling problem: Staffing ratios and nursing shortages. *Decision Sciences*, *37*(1), 39–70.

Xie, L., & Suhl, L. (2015). Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum*, *37*(1), 99–136.

Zuse Institute Berlin. (2014). *SCIP—Solving constraint integer programs*. http://scip.zib.de/