Solving Rostering Tasks as Constraint Optimization

Harald MEYER AUF'M HOFE

GWI-SIEDA GmbH
Richard-Wagner-Str. 91, D-67655 Kaiserslautern
Harald.Meyer@gwi-ag.com

Abstract. Based on experiences with the *ORBIS Dienstplan*-system [12] – a nurse rostering system that is currently used in about 60 German hospitals – this paper describes how to use constraint processing for automatic rostering. In practice, nurse rostering problems have many varying parameters: Working time accounts, demands on crew attendance, set of used shifts, working time models, etc. Hence, rostering requires a flexible formalism for representing the variants of the problem as well as a robust search procedure that is able to cope with all problem instances. The described approach differs in mainly two points from other constraint-based approaches [1, 22] to rostering.
On the one hand, the used constraint formalism allows the integration of fine-grained optimization tasks by fuzzy constraints, which a roster may partially satisfy and partially violate. Such constraints have been used to optimize the amount of working time and the presence on the ward. In contrast, traditional frameworks for constraint processing consider only crisp constraints which are either completely violated or satisfied. On the other hand, the described system uses an any-time algorithm to search for good rosters. The traditional constraint-based approach for solving optimization tasks is to use extensions of the branch&bound. Unfortunately, performance of tree search algorithms is very sensitive to even minor changes in the problem representation. *ORBIS Dienstplan* integrates the branch&bound into local search. The branch&bound is used to enable the optimization of more than one variable assignment within one improvement step. This search algorithm converges quickly on good rosters and, additionally, enables a more natural integration of user interaction.
Keywords: Employee Timetabling, Commercial Packages, Artificial Intelligence, Constraint Based Methods, Soft Computing, Local Search.

1 Introduction

The objective of rostering tasks is to label employees with a working (or idle) shift for each day of a certain period of time. Hence, these problems may be viewed as constraint satisfaction problems (CSP), that concern the assignment of values out of a known domain to a finite number of variables [19]. This article describes a system for solving nurse rostering problems of various kinds by constraint-based reasoning.

The nurse rostering system *ORBIS Dienstplan*[1] — a collaboration of the *GWI-SIEDA GmbH Kaiserslautern* with the *German Research Center for Artificial Intelligence (DFKI)* — is currently operational in about 60 hospitals and in fire departments

[1] The system is currently sold under different trade marks.

[12, 14]. This article describes representation and search in a prototype on constraint-based rostering that reflects the lessons learned from the commercial system. From the perspective of research on constraint reasoning, the system exemplifies an integration of branch&bound search into iterative search algorithms. Furthermore, this system demonstrates how to use soft and even fuzzy or non-crisp constraints from a constraint library that is inspired by common practice in *constraint logic programming (CLP)* [13].

Nurse rostering is a comparably hard rostering task since shift assignments are required to comply with many different constraints concerning rest time, preferred sequences of shifts, working time accounts, compensation of working shifts on weekends, the expected expenditure of work and, last but not least, employees' preferences. Several authors stressed the relevance of constraint-based methods to solve nurse rostering problems [1, 5, 22]. As Section 5 describes in detail, the work presented here differs in two major points from these approaches:

1. In several realistic situations, constraints on a roster cannot be satisfied completely. Consider, for instance, the case that some overtime work is necessary to guarantee a crew of appropriate size on the ward. In these cases, nurse rostering is rather a problem of constraint optimization than a problem of constraint satisfaction. Consequently, special techniques had to be invented to optimize compliance with *soft* and partly *fuzzy constraints*.
2. All approaches from the literature apply several heuristics which exploit characteristics of the current application. Such heuristics restrict the applicability of the resulting system to some special instances of nurse rostering where, for instance, only three shifts — early-morning shift, late shift, night shift — have to be assigned to the nurses. In contrast, *ORBIS Dienstplan* is applicable to arbitrary sets of shifts including longer day-turns and on-call duties. Additional parameters of the problem comprise working time accounts and a specification of a minimal respectively preferred size of crew attendance *that may differ from day to day*. Obviously, such flexibility requires much more generic search procedures.

Additionally, constraint-based representations of nurse rostering problems are quite large. Since one needs a variable for each nurse on each day, rostering problems typically comprise 150 to 1200 variables. The structure of the constraints between these variables characterize nurse rostering as a problem of high computational complexity.

This paper is organized as follows: The first technical section presents a representation of nurse rostering as constraint problem. This section starts with the description of a special formalization of constraint problems that covers optimization aspects as well as use of constraint libraries. Based on this formalism, a representation of rostering problems is given. The next section briefly describes the standard algorithms for searching constraint optimization problems: Iterative improvement and branch&bound search enhanced by constraint propagation. This section concludes with an integration of both search paradigms that may be considered as an iterative search allowing complex improvement steps. However, this integration of complex improvements raises the question of how to distinguish promising from useless improvement steps. The next section presents a heuristic that turned out to be successful on this task. Finally, a concluding section provides a comparison to related work: (1) The proposed formalization

of soft constraints relates for instance to the *valued constraint satisfaction (VCSP)* formalism [3, 16]. (2) Literature on compelling systems for constraint-based nurse rostering is used to point out the differences of this approach to the state of the art in this field.

2 Rostering Problems as Constraint Optimization

The basic framework for the proposed representation of rostering problems originates from the *constraint satisfaction problem (CSP)* whose basic ingredients are variables, a set of values called domain, and constraints that impose restrictions on how to assign values from the domain to variables. This section develops a formal representation of rostering problems in two steps:

(1) The traditional notion of constraint problems is extended to deal with optimization. Additionally, constraints are derived from a *constraint library*, i.e. the problem representation refers explicitly to a number of *constraint types* out of a library that provides elaborate methods of reasoning on constraints.

(2) In a second part, this section presents the required constraint types and the way they are used to represent rostering problems. Rosters are represented by an assignment of shifts to variables. Constraints represent demands on appropriate rosters.

2.1 Hierarchical Constraint Optimization Problems

Problems of constraint satisfaction or optimization concern the task of finding assignments of values from a certain domain D to variables from a set X. These assignments have to respect constraints from a set C. Specializations of this general task differ with respect to the representation of domains (numeric or finite set of symbols), constraints (binary or out of a constraint library), and the way that constraints assess assignments (hard or soft constraints). Rostering problems require a very general and therefore quite complex framework for the representation of constraint problems.

Assignments to the variables in X represent rosters. In order to distinguish better from worse rosters, the constraints of a rostering problem define a preference ordering $\succ$ among assignments (rosters) in such a way that $A_1 \succ A_2$ holds true for two assignments A_1 and A_2 iff A_2 complies better with the constraints than A_1. Each constraint $c \in C$ states a valuation of the assignments to some of the variables in X by use of a propagation method that is provided by a *constraint library*. The effect of such propagation methods can be described by a function φ that maps assignments to the local variables to a *degree of constraint violation*. A constraint violation of 0 says that the assignment satisfies the constraint whereas a degree of 1 indicates a total violation. Values in between these extremes represent a partial constraint violation. Thus, constraints of this framework are *non-crisp* or *fuzzy* in the sense that there is something in between complete violation and complete satisfaction.

Specifications of constraints comprise some more attributes in order to enhance applicability of propagation methods since the implementation of these methods often requires a large effort. A parameter p enables the user of the constraint library to adapt the used propagation method φ to the current optimization problem. Two additional

attributes of constraints specify which constraints are more important than others. The hierarchy level h of a constraint is a positive integer that represents the constraint's categorical importance: Each constraint of a more important hierarchy level is more important than all constraints in the levels of lower importance together. Traditionally, hierarchy level 0 is the most important hierarchy level comprising the mandatory constraints [4]. Additionally, each constraint exhibits a weight ω in order to state its gradual importance. This means that importance of constraints *of the same hierarchy level* grows with their weight. In contrast to hierarchy levels, satisfying more constraints of smaller weight may be preferred to the satisfaction of a single constraint of larger weight. These commitments result in the following definition of a constraint formalism.

A *hierarchical constraint optimization problem (HCOP)* is a tuple $\mathcal{P} = \langle X, D, C \rangle$ where X is a set of variables, D is a finite set of values, and C is a set of constraints.

A constraint $\langle h, \omega, \boldsymbol{x}, p, \varphi \rangle \in C$ is composed of the *hierarchy level* $h \in \mathbb{N}_0$, the constraint's weight $\omega \in \mathbb{R}^+$, the vector of n local variables $\boldsymbol{x} \in X^n$, the *parameter* $p \in P$, and the *degree of constraint violation* $\varphi : P \times D^n \to [0; 1]$.

Let A be an assignment of values from D to all variables in X and $A \downarrow \boldsymbol{x}$ result in a vector $\boldsymbol{d} \in D^n$ where $\boldsymbol{d}[i]$ is the value that is assigned to $\boldsymbol{x}[i]$ by A.

The weight of an assignment A in hierarchy level i is defined as

$$\Omega_h(A) = \sum_{\langle h, \omega\, \boldsymbol{x}, p, \varphi \rangle \in C} \omega \cdot \varphi(p, A \downarrow \boldsymbol{x}).$$

Assume A_1 and A_2 to denote two assignments of values from D to the variables in X, then the preference ordering $\succ$ — that distinguishes less from more preferred assignments — is defined as follows: If A_1 violates some mandatory constraints (hierarchy level 0) to a degree larger than 0 — equivalent to $\Omega_0(A_1) > 0$ — and A_2 satisfies all mandatory constraints — equivalent to $\Omega_0(A_2) = 0$ — then A_2 is better than A_1 or $A_1 \succ A_2$. $A_1 \succ A_2$ holds additionally true otherwise, iff both assignments satisfy the mandatory constraints and there is a hierarchy level i with $\Omega_i(A_1) > \Omega_i(A_2)$ and for all levels j with $1 \leq j < i$: $\Omega_j(A_1) = \Omega_j(A_2)$.

According to this definition, an assignment that satisfies all mandatory constraints is preferred to an assignment that violates some mandatory constraints. Two assignments that both violate mandatory constraints are considered to be equally bad. Preference referring to the soft constraints is determined according to the most important hierarchy level where one assignment violates less important constraints than the other assignment. Of course, HCOP is related to other generalizations of the standard CSP, namely lexicographic VCSP [3, 16]. Please refer to section 5 for further notes on this issue.

2.2 Rostering Problems as HCOP

The first task in representing real world problems as a constraint problem is to identify the variables and the domains. In our representation, a constraint variable is generated for each nurse on each day in the roster. The planning period is typically a month, i.e. 30 or 31 days. In the following, x_{ij} denotes the constraint variable of nurse i on day j. Variable x_{ij} is labeled with the shift that nurse i has to serve on day j. The descriptor of a shift is typically composed of a letter representing the shift type that is followed

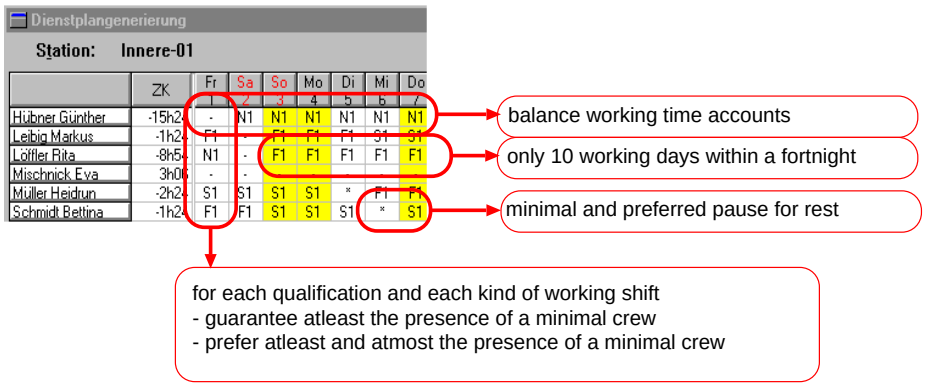

Fig. 1. *The constraints on a roster.*

by a number. Examples: F1 represents early-morning shift variant 1, S2 is the second variant of a late shift, N1 is a night shift. Often, longer day-turns like T3 are used in addition. The descriptors '$*$' and '–' denote idle shifts whereas holidays are indicated by UL. Shifts have three properties: Start time s, end time e, and working time w, where the latter represents the influence of the shift on working time accounts. The used set of working shifts varies often from hospital to hospital. Thus, the problem representation has to be able to deal with arbitrary sets of shifts. Fig. 1 shows a portion of a schedule to illustrate this representation.

Requirements on a roster are given as constraints which partly have a large number of local variables. Fig. 1 indicates instances of the main classes of constraints mapping local variables to textual descriptions of the constraints' extensions. The full representation of realistic problems comprises 300 to more than 2000 constraints. However, these constraints are weighted instances of a rather small set of constraint types. The following paragraphs describe these types more formally also illustrating their role in the representation of rostering problems.

Rest times: Two consecutive shifts of the same employee have to allow a minimal and a preferred time of rest. This demand can be ensured by two binary constraints between all consecutive shifts that may be implemented easily as so-called *extensional* constraints by use of propagation ext_n. This propagation simply receives the set of all consistent combinations of values as a parameter. The degree of constraint violation is defined as follows:

$$ext_n : 2^{D^n} \times D^n \longrightarrow 0;1 : ext_n(p, \boldsymbol{d}) = \begin{cases} 0 : \boldsymbol{d} \in p \\ 1 : \text{otherwise.} \end{cases}$$

Hence, a tuple $\boldsymbol{d}$ complies with the constraint if it is element of the extension p of the constraint.

The required constraints can be generated from this propagation defining parameter p, i.e. the extension. Usually, the minimal time of rest between shifts is 11 hours. Hence, on these constraints p comprises all pairs of shifts where the second shift starts more

than 11 hours after the first shift ends. Analogously, a preferred rest time of 16 hours can be specified.

Working time models: So-called working time models are preferred sequences of shifts that are stored in a database and range typically over two weeks. On the one hand, each nurse is preferred to work according to a certain working time model. The management of the ward assigns one working time model to each nurse in order to control which nurses are preferred to serve night shifts and which nurses alternate between different types of shifts. On the other hand, all working time models together represent a set of generally preferred shift sequences that nurses are accustomed to serve. As a consequence, the shift assignments to each nurse are also preferred to be consistent with an *arbitrary* working time model. Working time models differ strongly from hospital to hospital.

It is quite easy to state both demands by extensional constraints. For each nurse, consistency with the assigned working time model is represented by unary extensional constraints for each day whose extension comprises only the shift that the assigned working time model schedules on that day. Additionally, extensional constraints of 14 local variables have all specified working time models as extension.

Undesired sequences of shifts: There are also some sequences of shifts that shall be generally avoided. Examples are: Too long chains of night shifts and sequences of the form working shift, idle shift, working shift. Details on undesired sequences of shifts also vary from hospital to hospital.

The representation of such demands requires a constraint type whose parameter describes forbidden instead of allowed tuples of values. Let μ be a function mapping k-ary tuples to $\{0; 1\}$ with $0 < k \leq n$ where $\mu(\boldsymbol{d}) = 1$ iff tuple $\boldsymbol{d}$ should be avoided. Then the following degree of constraint violation states a penalty on any sequence of tuples that μ maps to 1:

$$avoid_{n,k} : avoid_{n,k}(\mu, \boldsymbol{d}) = \begin{cases} 1 : \exists i : \mu(\langle \boldsymbol{d}[i], \ldots \boldsymbol{d}[i+k-1] \rangle) = 1 \\ 0 : \text{otherwise.} \end{cases}$$

Different $\mu's$ can now be used to specify certain sequences of shifts that have to be avoided.

Ensure minimal crew: One of the fundamental demands on rosters is to guarantee a minimal crew on the ward. Example: On day 5 the roster has to schedule at least 1 early-morning shift F1 or F2 to nurses 2, 4, 7, or 8 since these nurses have a special qualification that is required at that time on the ward. The rostering system enables the management to formulate arbitrary demands of this form. Hence, a constraint type is needed that counts the occurrences of certain values in the assignment to variables and compares this to a goal sum g. Let μ be a function that maps shifts from the domain D to real numbers. Then, g and μ are parameters of a propagation method $\varphi = atleast_n$ implementing the following relation:

$$atleast_n\left(\langle g, \mu \rangle, \boldsymbol{d}\right) = \begin{cases} 0 : \sum_{i=1}^{n} \mu(\boldsymbol{d}[i]) \geq g \\ 1 : \text{otherwise.} \end{cases}$$

Thus, a constraint of this type with the local variables $\boldsymbol{x} = \langle x_{2,5}, x_{4,5}, x_{7,5}, x_{8,5} \rangle$ and parameter $p = \langle 1, \mu_{(F1,F2)} \rangle$ represents the demand of the example, where $\mu_{F1,F2}(d)$ is one iff d is either F1 or F2 and 0 otherwise.

Compensate work on weekends: Working shifts on Saturdays or Sundays need to be compensated within a fortnight. This demand is translated into a slightly tighter constraint that allows only 10 days of work within a fortnight. The type of this constraint is called $atmost_n$ and is defined analogously to $atleast_n$.

Prefer standard crew: The second demand on crew attendance is to prefer a standard crew that is often larger then the minimal crew. Example: On day 5 the roster should schedule preferably 3 early-morning shifts F1 or F2 to nurses 2, 4, 7, or 8. This means that also scheduling 2 early-morning shifts is preferred to scheduling only 1, etc. Propagation method $approx_n$ is used to represent such demands. Again, parameters are a function μ and a goal sum g. Additionally, this type receives an exponent e as parameter that will be used later on. In the meanwhile assume $e = 1$. The following definition of $approx_n$ shows that this propagation implements a fuzzy relation between assignments to the local variables:

$$approx_n\left(\langle g, \mu, e \rangle, \boldsymbol{d}\right) = \frac{\left| g^e - \left(\sum_{i=1}^{n} \mu(\boldsymbol{d}[i])\right)^e \right|}{n \cdot \max\{\mu(d)^e \mid d \in D\}}.$$

Hence, the degree of constraint violation grows with the difference between goal sum g and sum of the results of μ.

A constraint of the local variables $\boldsymbol{x} = \langle x_{2,5}, x_{4,5}, x_{7,5}, x_{8,5} \rangle$ can be used to represent the demand from the example. The parameter of the constraint is set to $p = \langle 3, \mu_{F1,F2}, 1 \rangle$. In contrast to constraints of the previously described types *ext*, *avoid*, *atmost*, and *atleast* constraint of type *approx* are fuzzy or non-crisp constraints: Most assignments to the local variables are neither completely satisfied nor fully violated. A roster assigning exactly 3 of the requested shifts to the affected nurses satisfies this constraint perfectly. Assigning only 2 of the requested shifts leads to a constraint violation of degree $\frac{1}{4}$. The minimal crew — where only one of the requested shifts gets assigned — causes a larger constraint violation of degree $\frac{1}{2}$, and so on.

Keep working time accounts in balance: Depending on time credits and working contracts, each nurse has to serve a certain amount of working time. The scheduled shifts have to approximate the available working time as well as possible. Propagation $approx_n$ is also appropriate to this task. In contrast to the management of crew assignment, on this demand the mapping w of shifts to working time is used as parameter μ. For each nurse i a constraint is required with $\boldsymbol{x} = \langle x_{i,1}, \ldots x_{i,31} \rangle$ as local variables. Parameter g of the constraint type reflects the amount of working time that nurse i has to serve on that month. Parameter e is set to 2 in order to distribute possibly necessary overtime work on as many shoulders as possible.

2.3 Use of Constraint Weights in Semi-Automatic Rostering

The previous section described local variables, constraints, and their parameters that are used to represent rostering problems. Furthermore, hierarchy levels and weights of the constraints need to be defined.

Moreover, complex rostering tasks can not be solved in an off-line manner since some demands on rosters will always be tacit, i.e. in the mind of responsible persons. Hence, rostering systems have to enable this responsible person to intervene into the rostering process. This section shows, that hierarchy levels and weights of the constraints relate closely to the organization of this dialog.

The system shall perform as follows. Once started, the system searches for a roster that complies well with the constraints as described in the previous section according to shifts, working time accounts, holidays, and working time models from a data base. Provided with information on the quality of the best yet found roster, the user of the system shall always be able to stop the search for better rosters and state new demands as unary, extensional constraints, e.g. defining some shifts of a certain nurse. Then, the rostering system shall be able to improve the best yet found roster respecting the newly introduced constraints until it is stopped again or the roster is perfect. This procedure enables the operator of the system to support the search for good rosters and to make tacit demands on the roster explicit. The portion of a roster in Fig. 1 is drawn from the presentation of an imperfect roster to the operator of the system. The shaded rows indicate days where the system yet failed to achieve a standard crew on the ward. Hence, the operator can easily recognize deficiencies of the best available roster.

As a consequence of user dialog, hierarchy level 0 of mandatory constraints only comprises the constraints that all rosters have to satisfy when presented to the operator. These are constraints on minimal resting times and approved holidays.

Hierarchy level 1 comprises the constraints that have been generated on previous interactions with the user. Hence, satisfying one interactively formulated constraint is more important than satisfying the complete initial problem representation — the operator of the system may override any non-mandatory part of the initial problem.

Hierarchy level 2 contains all demands on a legal roster: Minimal crew, at maximum 10 working days within a fortnight, and some mandatory bounds on the working time accounts.

Hierarchy level 3 is about preferred resting times. Hierarchy level 4 usually prefers the standard crew on the ward, whereas level 5 holds the constraints that try to keep working time accounts in balance. Sometimes, the levels 3 to 5 are put into another order if for instance too many time credits respectively time debts are known in advance — in this situation the constraints on working time accounts are more important than others.

Level 6 holds the constraints on working time models. The system enables nurses to state personal preferences in advance. These preferences are then translated into unary extensional constraints and put into hierarchy level 7. However, this feature of the rostering system is currently more important on selling the system than on running the system, i.e. many hospitals want to have the opportunity to deal with personal preferences of the employees but currently most of the hospitals do not use this feature.

Within the hierarchy levels, the weights of the constraints are usually set to 1, i.e. the solver tries to satisfy as many demands as possible.

This section shows that representing rostering as a constraint problem leads to a formalization that allows user interaction and supports problem reformulation on adding new constraints. Especially hierarchy levels proved to be useful in adopting constraint models on rostering to the current situation. Although the formal definition of HCOPs is rather complex, users of the system gain surprisingly fast a sufficient understanding of the semantics of hierarchy levels and constraint weights.

2.4 Comparison to Standard Constraint Logic Programming

On a first glance, especially the crisp constraint types *ext*, *avoid*, *atmost*, and *atleast* seem to be quite similar to constraints which are provided by programming languages like CHIP [6] or ECl^{i}PS^e [20] from the field of constraint logic programming (CLP). However, Section 3.2 shows that constraints of the proposed kind allows the combination of branch&bound search á la partial constraint satisfaction [9] with propagation procedures from CLP. So, the algorithms for constraint propagation are quite similar but, in contrast to CLP, constraint propagation does not only prune the domains of unlabeled variables.

Moreover, non-crisp constraints like the ones of type *approx* are unknown to CLP. Similar dependencies are represented by higher-order predicate `minimize(G(X), f(X))`. This predicate allows only that substitutions to the variables in `X` that are minimal referring to the result of function `f(X)` among the substitutions that satisfy the goals `G(X)`. Fages presented a declarative semantics of this predicate [8] and some examples for its non-intuitive meaning. The implementation of this constraint runs a branch&bound search on the part of the problem as described by `G(X)` and `f(X)` without taking any advantage from constraint propagation as described in Section 3.4 of this paper.

3 Mixed-Paradigm Search

Several search algorithms have been proposed to solve constraint optimization problems namely iterative improvement and branch&bound search. The iteration of user dialogue and solving process as described in section 2.3 suggests to use an iterative repair algorithm. However, as pointed out below, standard algorithms in this field exhibit some significant drawbacks that are relevant to rostering problems. Hence, some work on solving constraint problems in general had been necessary to implement the described nurse rostering system. This system uses an integration of both, iterative improvement and branch&bound.

3.1 Iterative Improvement

Fig 2 shows the structure of algorithms on iterative improvement. These algorithms start with computing an initial assignment to all variables. Then, a loop of improvement steps follows. The standard algorithm in this field is the *MinConWalk* [21] that chooses

1: compute an initial assignment A to all variables in X.
2: **loop**
3: set $X' \subseteq X$ to hold a region where A possibly is suboptimal.
4: **if** $X' = \emptyset$ **then break**, **end if**
5: change assignments of A to the variables in X'.
6: **end loop**
7: **return** A

Fig. 2. *Structure of algorithms on iterative improvement.*

in line 3 a single variable at random. The operation in line 5 depends on a so-called *walk probability* p. With probability $1 - p$, line 5 assigns an optimal value to the variable in X'. In order to escape from local minimal, line 5 assigns with probability of p a randomly chosen value. Although these algorithms have been pretty successful on several kinds of constraint problems, they perform not too well on problems implying constraints of many local variables that are hard to satisfy.

Consider Fig. 3 as an example. The box above (situation 1) presents a portion of a roster comprising shifts of the nurses *A* and *B*. The area within the dotted box complies with working time models concerning early-morning shifts. However, there is a problem with the minimal resting time for nurse *A* between the days 0 and 1: The morning shift F1 starts too early after late shift S1. The box below (situation 2) presents an improvement to this roster: It complies with the same working time models, working time and crew attendance is the same as in the roster above. Nevertheless, the previously violated constraint on times of rest is now satisfied. This improvement has been achieved changing assignments to 8 variables all at once, but each change of less than 8 assignment makes the roster worse. Note that compliance with working time models, although the corresponding constraints form a deeper part of the hierarchy, are very important since they provide the only *sufficient* conditions on acceptable sequences of shifts (whereas constraints on number of working days and times of rest provide only *necessary* conditions). Hence, the rostering system is required to satisfy working time models well.

Standard algorithms on iterative improvement can, in theory, deal with this situation but the probability of finding the described improvement is very low. Typical extensions of iterative search like tabu lists [18] fail to increase this probability significantly. Such situations obviously require the availability of more complex improvement steps where the assignments to more than one variable get changed. The branch&bound algorithm is appropriate to conduct such complex improvement steps.

3.2 Extending Branch&Bound Search by Constraint Propagation

Propagation of soft constraints provides an optimistic estimate on the quality of a solution that can be found in the current branch of the search tree [9]. For each labeling $x \leftarrow d$ of a variable $x \in X$ with a value $d \in D$, tree search algorithms with propagation of soft constraints maintain a data structure $conflicts[x][d]$ to collect a measure for the importance of the constraints that are violated by all leaves in the current branch of the

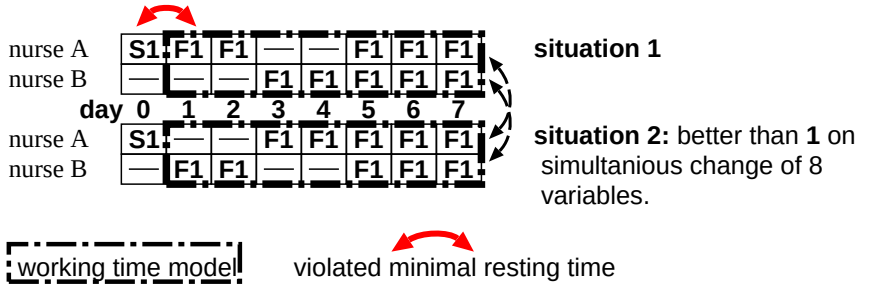

Fig. 3. *Example for large bad regions caused by working time models.*

search tree that label x with d. The term *constraint propagation* denotes algorithms that detect new conflicts to be stored in $conflicts[x][d]$ referring to the properties of the propagated constraint and previously detected conflicts. Estimate $conflicts[x][d]$ can then be used by the A*-heuristic [15] to guide depth-first search. Additionally, adoptions of the well-known *minimum remaining values (MRV)* heuristic to constraint hierarchies are known to improve the order of the variables in the search tree [11, 13].

This section presents a new integration of constraint propagation into branch&bound search with two major characteristics: (1) The search algorithm is able to integrate special algorithms for the efficient propagation of non-binary constraints which may have non-crisp extensions as exemplified by constraint type *approx*. (2) The algorithm uses fuzzy sets [23] on the set of constraints C to represent estimate $conflicts[x][d]$. These sets map a certain degree of membership to each constraint c. During search, this degree of membership corresponds with the minimal degree of violating constraint c in the current branch of the search tree assigning d to x.

More formally: A fuzzy set on values from the crisp set S is a tuple $\tilde{S} = \langle \mu, S \rangle$ where μ is a mapping from S to numbers from the interval $[0; 1]$ that is called *membership function*. The membership function assigns a membership degree to each value $s \in S$ where $\mu(s) = 1$ designates full members and $\mu(s) = 0$ characterizes non-members of the fuzzy set $\tilde{S}$. Thus, a crisp set S can be treated as a fuzzy set assigning membership 1 to all the elements and membership 0 to all non-elements. The empty set $\emptyset$ is equivalent to a fuzzy set assigning only membership 0. Intersection $\sqcap$ and union $\sqcup$ of fuzzy sets are defined as follows: $\langle \mu_1, S_1 \rangle \sqcap \langle \mu_2, S_2 \rangle = \langle \mu_3, S_1 \cap S_2 \rangle$ where $\mu_3(c) = \min\{1, \mu_1(s), \mu_2(s)\}$ and $\langle \mu_1, S_1 \rangle \sqcup \langle \mu_2, S_2 \rangle = \langle \mu_4, S_1 \cup S_2 \rangle$ with $\mu_4(c) = \max\{0, \mu_1(s), \mu_2(s)\}$.

Fig. 4 presents a branch&bound algorithm that uses fuzzy sets on the constraints C to record conflicts. Assume assignment A to label each variable in X with a value from D and fuzzy set $\overline{C}(A)$ is intended to provide a measure for the conflicts of assignment A. For each constraint $c = \langle h, \omega, \boldsymbol{x}, p, \varphi \rangle$, $\varphi(p, A \downarrow \boldsymbol{x})$ is A's degree of violating c. Thus, $\varphi(p, A \downarrow \boldsymbol{x})$ provides a measure for A's conflict with constraint c that can serve as a membership in a fuzzy set of constraints reflecting A's conflicts with all constraints: $\overline{C}(A) = \langle \mu, C \rangle$ with $\mu(\langle h, \omega, \boldsymbol{x}, p, \varphi \rangle) = \varphi(p, A \downarrow \boldsymbol{x})$.

Measuring conflicts in terms of fuzzy sets on constraints requires a notion of preference that complies with the preference among assignments. Thus, the preference ordering $\succ$ of Section 2.1 needs to be lifted to fuzzy sets of constraints in such a way that for all assignments A_1 and A_2 of domain values to variables, $A_1 \succ A_2$ is equivalent to $\overline{C}(A_1) \succ \overline{C}(A_2)$. Again, a Ω_i is defined to sum up the weights in hierarchy level i: $\Omega_i(\langle \mu, C' \rangle) = \sum_{c \in C'} \omega \cdot \mu(c)$ with $c = \langle h, \omega, \boldsymbol{x}, p, \varphi \rangle$. This sum of weights within a hierarchy level is used to lift the preference ordering on two fuzzy sets of constraints $\tilde{C}_1$ and $\tilde{C}_2$ as follows:

$$\begin{aligned}\tilde{C}_1 \succ \tilde{C}_2 :\Longleftrightarrow & \Big(\Omega_0(\tilde{C}_1) > 0 \wedge \Omega_0(\tilde{C}_2) = 0\Big) \vee \\ & \Big(\big(\Omega_0(\tilde{C}_1) = \Omega_0(\tilde{C}_2) = 0\big) \wedge \\ & \quad \big(\exists i > 0 : \Omega_i(\tilde{C}_1) > \Omega_i(\tilde{C}_2)\big) \wedge \\ & \quad \big(\forall j : 1 \leq j < i \Rightarrow \Omega_j(\tilde{C}_1) = \Omega_j(\tilde{C}_2)\big)\Big).\end{aligned}$$

The proposed version of the branch&bound as presented in Fig. 4 exploits three advantages of fuzzy sets of this kind as representation of detected conflicts:

1. Assume that the two fuzzy sets on constraints $\tilde{\delta_1}$ and $\tilde{\delta_2}$ hold conflicts of a certain assignment with the constraints. Then, $\tilde{\delta_1} \sqcup \tilde{\delta_2}$ holds also valid conflicts of this assignment, since fuzzy set union simply collects the largest degrees of constraint violation that have been proven previously and stored either in $\tilde{\delta_1}$ or in $\tilde{\delta_2}$.
2. Additionally, fuzzy set union is an idempotent operation, i.e. $\tilde{\delta} \sqcup \tilde{\delta} = \tilde{\delta}$ for any fuzzy set of conflicts $\tilde{\delta}$. This means that conflicts are counted only once even if they have been added twice or more.
3. Assume, that assignment S is the best currently known solution to the problem and fuzzy set $\tilde{\delta}$ comprises conflicts of the currently explored partial assignment. Then, $\tilde{\beta} = \overline{C}(S)$ may serve as a bound, i.e. the algorithm can backtrack without loosing solutions better than S as soon as $\tilde{\delta} \succ \tilde{\beta}$ holds true.

Algorithm BB-SEARCH conducts a depth-first tree search of the branch in the search tree that is described by partial assignment A, where distance $\tilde{\delta}$ is a fuzzy set of constraints comprising the previously detected conflicts of A and $X' \subseteq X$ is the set of variables that shall be labeled by the algorithm. S is the best yet found solution and bound $\tilde{\beta}$ contains the conflicts of this solution. The algorithm results into the best assignment S to the variables in X' according to preference ordering $\succ$ or, equivalently, the least important conflicts $\overline{C}(S)$.

Function BB-START simply starts BB-SEARCH with initial arguments: S, A, and distance $\tilde{\delta}$ are empty. Bound $\tilde{\beta}$ has a maximal value. All variables shall be labeled. Array $conflicts$ holds the empty set for each labeling since constraint propagation has not yet proven any conflict. So, let us go back to BB-SEARCH.

Line 1 returns S if the distance $\tilde{\delta}$ is not better than bound $\tilde{\beta}$, i.e. the currently explored branch of the search tree does not contain any assignment that is better than S. Line 2 says that otherwise the currently explored assignment A is an improvement of S if all variables are labeled by A. Thus, A is a new solution. Line 3 performs an update of the $conflicts$. This is necessary because $\tilde{\delta}$ and the entries of $conflicts$ both hold valid

BB-START(X, D, C)

1: Store problem variables X, domain D, and constraints C as global variables.
2: Generate a 2-dimensional array $conflicts[X][D]$ and initialize each entry with the empty set.
3: **return** BB-SEARCH$(\emptyset, C, \emptyset, \emptyset, conflicts, X)$.

BB-SEARCH$(S, \tilde{\beta}, A, \tilde{\delta}, conflicts, X')$

1: **if** $\tilde{\beta} \not\succ \tilde{\delta}$ **then return** $\langle S, \tilde{\beta} \rangle$ **end if**.
2: **if** $X' = \emptyset$ **then return** $\langle A, \tilde{\delta} \rangle$ **end if**.
3: **for all** $x \in X'$ and $d \in D$ **do** $conflicts[x][d] \leftarrow conflicts[x][d] \sqcup \tilde{\delta}$. **end for**
4: Choose a variable $x \in X'$ according to a specialized MRV heuristic.
5: Copy the $conflicts[x]$ to the new array $conflicts'$.
6: **for all** $d \in D$ with $conflicts[x][d] \not\succ \tilde{\beta}$ choosing d's with smaller conflicts first **do**
7: **for all** $d' \in D$ with $d' \neq d$ **do** $conflicts[x][d] \leftarrow C$. **end for**
8: **for all** $c = \langle h, \omega, \boldsymbol{x}, p, \varphi \rangle \in C$ where x is a local variable **do**
9: Update $conflicts$ according to propagation of c with an algorithm corresponding with φ with the arguments $(c, \boldsymbol{x}, \tilde{\beta}, p, conflicts')$.
10: **end for**
11: **if** for all variables x' there is a $d' \in D$ with $\tilde{\beta} \succ conflicts[x'][d']$ **then**
12: $\langle S, \tilde{\beta} \rangle \leftarrow$ BB-SEARCH$(S, \tilde{\beta}, A \cup \{x \leftarrow d\}, conflicts[x][d], conflicts, X' \setminus \{x\})$.
13: **if** $\tilde{\beta} = \emptyset$ **then return** $\langle S, \emptyset \rangle$, **end if**.
14: **end if**
15: $conflicts[x] \leftarrow conflicts'$.
16: **end for**
17: **return** $\langle S, \tilde{\beta} \rangle$.

Fig. 4. *Branch&bound for solving HCOPs.*

conflicts and from here on $conflicts[x][d]$ is required to hold all known conflicts that result from adding assignment $x \leftarrow d$ to A. These conflicts are typically needed by the heuristics for dynamic variable reordering in line 4 that determine which variable to label next. Heuristic *minimal remaining values (MRV)* [2] (that is sometimes called *first fail*) is one of the most successful heuristics that is applicable here. It chooses the variable where the number of values which are not proven to be inconsistent with mandatory constraints is as small as possible[2]. In this implementation this is the variable x where a minimal number of values d do not have a mandatory constraint in $conflicts[x][d]$ with a membership larger 0.

Branching is done in the loop running over the lines 6 to 15. Note, that $conflicts[x][d]$ now holds all conflicts which are known to result from assigning d to x. Solutions to the problem are required to be better than bound $\tilde{\beta}$. Only that values with smaller conflicts are *admissible*, i.e. need to be considered for branching. Additionally, $conflicts[x][d]$ allows to follow the A*-heuristic [15]: Try the assignments with the least important conflicts first.

Line 7 simulates assigning value d to variable x. All other values are marked with a maximal set of conflicts. After selecting value d for variable x, BB-SEARCH conducts

[2] Refer to [11, 13] for extensions of this heuristic that consider the complete constraint hierarchy.

an *extended forward checking* of soft constraints calling propagation methods. This procedure is related to *forward checking* on soft constraints [9], but this algorithm calls propagation without regarding the number of variables that have not yet been labeled.

Propagation of constraint $c = \langle h, \omega, \boldsymbol{x}, p, \varphi \rangle$ adds c with a certain membership to the $conflicts$ of the local variables. The membership of constraint c in $conflicts[x][d]$ is intended to be an estimate of the effect of adding $x \leftarrow d$ to the currently explored partial assignment A. The rostering system uses a propagation rule for inferring this membership that allows the application of algorithms which are similar to the ones that are used in CLP[3]. The concrete algorithm is chosen according to the constraint type φ, but all these procedures obey the same specification. Propagation of c looks for the smallest degree of violating c that is caused by assignments to the local variables of constraint c that exclusively consider *admissible* labelings. More formally, assume that propagation of c has to determine c's new membership to $conflicts[x][d]$ where $x = \boldsymbol{x}[i]$. D^n is the set of vectors representing assignments to the n local variables of c. Now, filter out any vector $\boldsymbol{d}$ with $\boldsymbol{d}[i] \neq d$. These vectors represent assignments to the local variables that are not relevant to labeling $x \leftarrow d$. Furthermore, prune all vectors $\boldsymbol{d}$ concerning a non-admissible labeling — there is a j with $1 \leq j \leq n$ and $conflicts[\boldsymbol{x}[j]][\boldsymbol{d}[j]] \succ \tilde{\beta}$. These assignments consider labelings that cause too many conflicts in order to be part of any solution. Let $\boldsymbol{D}$ denote the remainder. Then, $\mu = \min\{\varphi(p, \boldsymbol{d}) \mid \boldsymbol{d} \in \boldsymbol{D}\}$ is the minimal degree of violating c that is a consequence of assigning d to x. So, c is added with this membership to $conflicts[x][d]$.

Constraint propagation adds stronger memberships to the $conflicts$. This may lead to the case that for some labelings $x' \leftarrow d'$, $conflicts[x'][d']$ becomes larger than bound $\tilde{\beta}$. As a consequence, search in deeper branches of the search tree will ignore this labeling since it is not admissible. Line 11 tests for the case that a variable x' does not have any admissible labelings. In this case, there is no need to explore the current branch of the search tree because it does not comprise improvements to S.

Otherwise, branching is done in line 12 by a recursive call of BB-SEARCH where A is extended by the new labeling, $\tilde{\delta}$ is replaced by the conflicts caused by the new labeling and x is removed from the variables to be labeled. Note, that the presentation of Fig. 4 assumes a call by value that copies the arguments. Thus, the recursive call of BB-SEARCH returns the best labeling that has been found in the deeper branches of the search tree without touching the current data on conflicts. This can be considered as an implementation of the backmarking strategy [9, 10]. Finally, line 15 undoes the selection of d in x.

3.3 A Very Small Example

Reconsider the small example of Fig. 3. This problem concerns two nurses on 8 days and, consequently, 16 variables. The representation as described in Section 2.2 requires more than 80 constraints. To cut a long story short, this section only refers to the constraints on minimal pause of rest, some constraints on working time models, and 7 constraints $atleast_1$ to $atleast_7$ enforcing at least one morning shift F1 on each of

[3] The propagation rule of the max-min-algorithm [17] does unfortunately not comply with algorithms for constraint propagation from CLP [13].

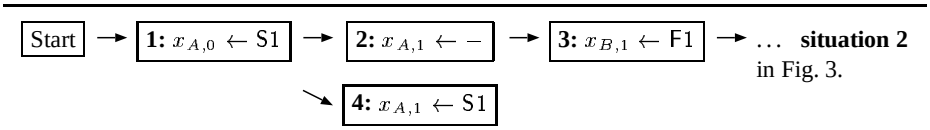

Fig. 5. A possible search tree for the example of Fig. 3.

the days 1 to 7. Let $x_{A,3}$ denote the constraint variable for the shift of nurse A on day 3. For each pair of variables $x_{N,i}, x_{N,i+1}$ concerning consecutive shifts of the same nurse N, a constraint $pause_{N,i}$ allows the assignment of $(\mathsf{F1}, \mathsf{F1})$, $(\mathsf{F1}, \mathsf{S1})$, $(\mathsf{S1}, \mathsf{S1})$, or all combinations with the idle shift $-$. Two constraints $wmod_A$ and $wmod_B$ affecting the variables $x_{A,1}, \ldots, x_{A,7}$ and $x_{B,1}, \ldots, x_{B,7}$ respectively both allow the following two sequences of shifts: $(\mathsf{F1}, \mathsf{F1}, -, -, \mathsf{F1}, \mathsf{F1}, \mathsf{F1})$ and $(-, -, \mathsf{F1}, \mathsf{F1}, \mathsf{F1}, \mathsf{F1}, \mathsf{F1})$. Furthermore, assume that $x_{A,0}$ and $x_{B,0}$ hold the carry-over from the previous plan. So, $x_{A,0} = \mathsf{S1}$ and $x_{B,0} = -$ is fixed by mandatory hard constraints.

Fig. 5 sketches the search tree of this problem. After adding labeling 1 assigning S1 to $x_{A,0}$, propagation of $pause_{A,0}$ adds $pause_{A,0}$ as a full member to $conflicts[x_{A,1}][\mathsf{F1}]$. The entries $conflicts[x_{A,1}][\mathsf{S1}]$ and $conflicts[x_{A,1}][-]$ remain empty. Labeling 2 adds $atleast_1$ as a full member to $conflicts[x_{B,1}][-]$ and $conflicts[x_{B,1}][\mathsf{S1}]$, since assigning F1 to $x_{B,1}$ is the only admissible opportunity to satisfy this constraint. Additionally, constraint $wmod_A$ is added as full member to the conflicts of all labelings concerning shifts of nurse A that do not comply with sequence $(-, -, \mathsf{F1}, \mathsf{F1}, \mathsf{F1}, \mathsf{F1}, \mathsf{F1})$. Variable $x_{B,1}$ has the smallest domain, now. Labeling 3 is the best alternative here. Propagation of $wmod_B$ adds a conflict to all labelings concerning shifts of nurse B that do not comply with sequence $(\mathsf{F1}, \mathsf{F1}, -, -, \mathsf{F1}, \mathsf{F1}, \mathsf{F1})$. At this point, the labelings with empty entries in array $conflicts$ characterize exactly *"situation 2"* in Fig. 3. So, search proceeds quickly to this solution that does not exhibit any conflict. A solution without conflicts is always optimal so line 13 of algorithm BB-SEARCH terminates search immediately. However, proving optimality of a known solution is often harder than finding it. In order to investigate the benefits of backmarking in proving optimality, assume that line 13 has been missed in algorithm BB-SEARCH.

Finding a solution lowers bound $\tilde{\beta}$ and turns all labelings with larger conflicts into non-admissible ones. Consequently, all variables except $x_{A,1}$ have only one admissible label — the one that is concerned by the solution. Remember, that the backmarking strategy records all inferred conflicts. Thus, backtracking is done without any constraint propagation until changing the label of $x_{A,1}$. Here, labeling 4 is the only admissible alternative. However, propagation of $wmod_A$ reveals that this constraint cannot be satisfied since S1 is not concerned by any working time model. As a consequence, propagation of $wmod_A$ adds this constraint to any entry in $conflicts$ referring to a labeling of one of $wmod_A$'s local variables turning these labelings into non-admissible ones. This is a reason for an immediate backtracking — the whole search tree has been traversed.

ATLEAST $(c, \boldsymbol{x}, \tilde{\beta}, \langle g, \mu \rangle, conflict)$

1: Initialize each entry of $\hat{\mu}[1 \ldots n]$ with 0 where n is the number of entries in $\boldsymbol{x}$.
2: $\hat{\mu}_\Sigma \leftarrow 0$.
3: **for all** $i \in \{1 \ldots n\}$ **do**
4: **for all** $d \in D$ with $\tilde{\beta} \succ conflicts[\boldsymbol{x}[i]][d]$ **do**
5: **if** $\mu(d) > \hat{\mu}[i]$ **then** $\hat{\mu}[i] \leftarrow \mu(d)$ **end if**
6: $\hat{\mu}_\Sigma \leftarrow \hat{\mu}_\Sigma + \hat{\mu}[i]$.
7: **end for**
8: **end for**
9: **for all** $i \in \{1 \ldots n\}$ **do**
10: **for all** $d \in D$ with $\tilde{\beta} \succ conflicts[\boldsymbol{x}[i]][d]$ **do**
11: **if** $\mu(d) + \hat{\mu}_\Sigma - \hat{\mu}[i] < g$ **then**
12: $conflicts[\boldsymbol{x}[i]][d] \leftarrow conflicts[\boldsymbol{x}[i]][d] \sqcup \langle 1, \{c\} \rangle$.
13: **end if**
14: **end for**
15: **end for**
16: **return** $conflicts$.

Fig. 6. *Propagating $atleast_n$ constraints.*

3.4 How to Propagate Soft Constraints

The above-mentioned extension of branch&bound search by constraint propagation presented a specification of the propagation rule but neglected the issue of implementation. Most of the literature on constraint processing is on binary constraint that can be propagated efficiently enumerating all assignments to the local variables. However, rostering applications exhibit constraints that affect a large number of local variables. These constraints require, thus, special purpose algorithms for propagation. This section presents two examples.

The $atleast_n$ constraint has been invented in section 2.2 to formalize least requirements on the presence of the crew. It is similar to the one-dimensional cardinality constraints which are provided by many CLP languages. Beside the important role of this constraint in the nurse rostering domain, it is cited here to demonstrate how to use propagation methods from CLP to derive soft conflicts.

Remember: $\mu(d)$ defines a weight for each value $d \in D$. Within each complying assignment to the local variables of $atleast_n$-constraint c, the sum of these weights has to be larger than or equal to the goal sum g. Propagating $atleast_n$-constraints means: Look for values d where $\mu(d)$ is too small in order to allow satisfaction of the constraint. Fig. 6 presents an implementation of a corresponding propagation method. The lines 1 to 8 determine for each local variable $\boldsymbol{x}[i]$ of the constraint the largest $\mu(\cdot)$ of admissible values. Additionally, the sum of these largest weights is stored. This information is used in line 11 to determine whether constraint c can be satisfied assigning a certain value to a certain local variable. This constraint is crisp since it is either satisfied or violated. The next example is on the propagation of fuzzy constraints.

APPROX $(c, \boldsymbol{x}, \tilde{\beta}, \langle g, \mu, e\rangle, conflict)$

1: Initialize each entry of $\hat{\mu}[1 \ldots n]$ with 0 and $\check{\mu}[1 \ldots n]$ with -1.
2: $\hat{\mu}_\Sigma \leftarrow 0.\check{\mu}_\Sigma \leftarrow 0$.
3: **for all** $i \in \{1 \ldots n\}$ **do**
4: **for all** $d \in D$ with $\tilde{\beta} \succ conflicts[\boldsymbol{x}[i]][d]$ **do**
5: **if** $\mu(d) > \hat{\mu}[i]$ **then**$\hat{\mu}[i] \leftarrow \mu(d)$ **end if**
6: **if** $\check{\mu}[i] = -1 \vee \check{\mu}[i] > \mu(d)$ **then**$\check{\mu}[i] \leftarrow \mu(d)$ **end if**
7: $\hat{\mu}_\Sigma \leftarrow \hat{\mu}_\Sigma + \hat{\mu}[i].\check{\mu}_\Sigma \leftarrow \check{\mu}_\Sigma + \check{\mu}[i]$.
8: **end for**
9: **end for**
10: **for all** $i \in \{1 \ldots n\}$ **do**
11: **for all** $d \in D$ with $\tilde{\beta} \succ conflicts[\boldsymbol{x}[i]][d]$ **do**
12: **if** $\mu(d) + \hat{\mu}_\Sigma - \hat{\mu}[i] < g$ **then**
13: $conflicts[\boldsymbol{x}[i]][d] \leftarrow conflicts[\boldsymbol{x}[i]][d] \sqcup \left\langle \frac{(\mu(d)+\hat{\mu}_\Sigma-\hat{\mu}[i])^e - g^e}{(\max\{\mu(d)^e \mid d \in D\}}, \{c\} \right\rangle$.
14: **else if** $\mu(d) + \check{\mu}_\Sigma - \check{\mu}[i] > g$ **then**
15: $conflicts[\boldsymbol{x}[i]][d] \leftarrow conflicts[\boldsymbol{x}[i]][d] \sqcup \left\langle \frac{g^e - (\mu(d)+\hat{\mu}_\Sigma-\hat{\mu}[i])^e}{\max\{\mu(d)^e \mid d \in D\}}, \{c\} \right\rangle$.
16: **end if**
17: **end for**
18: **end for**
19: **return** $conflicts$.

Fig. 7. *Propagating $approx_n$ constraints.*

Fig. 7 shows the propagation of $approx_n$-constraints. As section 2.2 mentioned, a tuple $\boldsymbol{d}$ violates such constraints with a degree growing with $|g - \sum_{i=1}^{n}[\mu(\boldsymbol{d}[i]]|$. Additionally, an exponent e may specify that the geometric distance is used instead of the arithmetic distance. Propagation of these constraints starts with finding for each variable $\boldsymbol{x}[i]$ the maximal weight $\hat{\mu}[i]$ and the minimal weight $\check{\mu}[i]$ of an admissible value. Additionally, the sum of the maximal and minimal weights gets stored. This information is used in the lines 10 to 18 to decide for each admissible assignment to a local variable, whether goal g can still be reached. If the sum of weights of assigned values will be too small, then the degree of constraint violation is determined due to $\hat{\mu}$ (line 13). Otherwise, the minimal weights $\check{\mu}$ are used to determine the degree of constraint violation (line 15).

The effort for propagating both constraints grows only linear with the number of local variables n. Hence, these propagation methods are also appropriate to constraints affecting many local variables as they appear in rostering problems.

3.5 Branch&Bound as Improvement Step

The previous section stressed that both, iterative search and branch&bound, have their advantages and drawbacks. An integration of both algorithms according to Fig. 8 can inherit the advantages of both search paradigms. The basic idea of this algorithm is to follow the main procedure in iterative algorithms but enable more complex improvement steps by use of the branch&bound. Hence, Fig. 8 details the algorithm of

ITERATIVESEARCH(X, D, C)

1: $\tilde{\beta} = C$. $S \leftarrow \emptyset$.
2: Set initial solution S according to the working time models that have been assigned to each employee by the management.
3: Generate $conflicts[X][D]$ with $conflicts[x][d] = \emptyset$ forall $x \in X$ and $d \in D$.
4: **loop**
5: $X' \leftarrow$ CHOOSEBADREGION$(X, D, C, S, \tilde{\beta})$.
6: **if** $X' = \emptyset$ **then break**, **end if**
7: $\tilde{\delta} \leftarrow \tilde{\beta}$. Load A with all labelings in S that do not affect variables in X'.
8: **for all** $x \in X', d \in D$ **do** set $conflicts[x][d]$ to the empty set if S assigns d to x and to C otherwise. **end for**.
9: **for all** $c \in C$ with c has a local variable in X' **do**
10: $\tilde{\delta} \leftarrow \tilde{\delta} \sqcap (C \setminus \{c\}$. Set $conflicts$ due to propagation of c.
11: **end for**
12: $\langle S, \tilde{\beta} \rangle \leftarrow$ BB-SEARCH$(S, \tilde{\beta}, A, \tilde{\delta}, conflicts, X')$.
13: **end loop**
14: **return** $\langle S, \tilde{\beta} \rangle$

Fig. 8. *Iterative repair using branch&bound and constraint propagation.*

Fig. 2: The initial assignment to the variables is generated in the line 2 according to the working time models that have been assigned to the employees (refer to Section 2.2).

The loop of improvement steps ranges over the lines 4 to line 13. A function CHOOSEBADREGION is assumed to point at a region $X' \subseteq X$ in the current solution S where changes in the current assignments are likely to result into an improved solution. The algorithm terminates in line 6 if this procedure can not find any region in the solution that is suboptimal. Otherwise, the branch&bound is called to find an optimal assignment to the variables in X'. Therefore, the distance is initialized to comprise all conflicts of the persistent assignments in S to the variables in $X \setminus X'$. Additionally, $conflicts$ is initialized to hold all conflicts between variables in X' with the persistent assignments to the variables in $X \setminus X'$. After these preparation, the call of BB-SEARCH in line 12 is able to find improvements to the variables in X' if this is possible.

This procedure may be stopped after each improvement step by user input. Hence, this algorithm supports user interaction as described in section 2.3. Moreover, improvement steps are in contrast to standard algorithms not restricted to affect only the assignment to one variable — assignments to many variables may get changed depending on the output of function CHOOSEBADREGION. However, the implementation of this function is the main problem of this approach. As the next section shows, the *ORBIS Dienstplan* system uses heuristics for this purpose which base upon some general assumptions on characteristics of rostering problems.

4 How to Find Bad Regions in a Roster

The currently available *ORBIS Dienstplan* system uses heuristics to detect the shift assignments in a roster that are responsible for deficiencies. The implementation of these

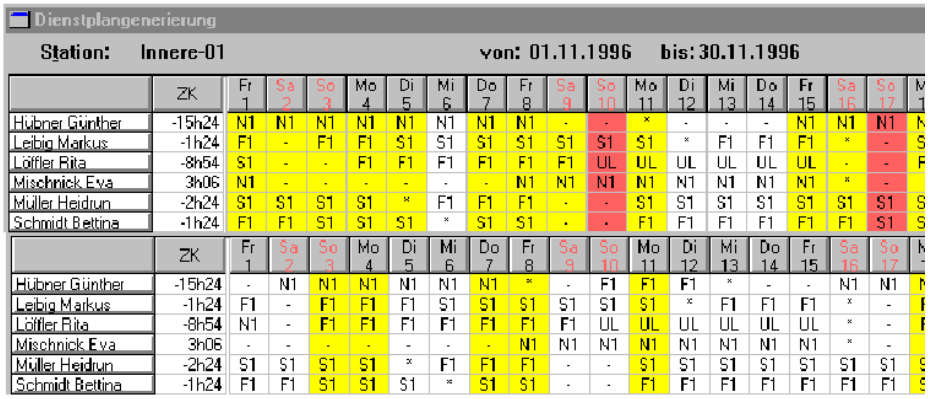

Fig. 9. Initial (above) and improved (below) roster.

heuristics is provided by the function CHOOSEBADREGION that is used in algorithm ITERATIVESEARCH. This section provides a description of the heuristic to achieve the minimal and preferred crew on the ward in order to illustrate this procedure. This heuristic is called whenever a constraint on crew attendance is more or less violated.

Fig. 9 presents (intermediate) results of the *ORBIS Dienstplan* system to a small sample problem that considers 6 nurses (186 variables and about 600 constraints). The roster above represents the initial assignment of shifts before any step of repair. The roster below is an acceptable solution which has been computed conducting several repair steps. Light shadings represent suboptimal but acceptable deficiencies in crew attendance. Dark shadings represent unacceptable deficiencies in crew attendance.

First of all, consider the problem on Sunday the 17th where an early-morning shift (F1) is missing and two late shifts (S1) have been scheduled but we prefer to have only one on that day. On such easy problems, changing only a single assignment of a shift suffices to achieve an improvement: Assigning an early-morning shift instead of a late shift to Bettina Schmidt on that day satisfies both previously violated constraints.

On Sunday the 10th, the schedule exhibits a worse problem — again an early-morning shift is missing. In such situations, at least two assignments of shifts have to be changed: A currently idle person has to serve an early-morning shift on Sunday the 10th and this additional working time has to be compensated on another day because of working time restrictions. Procedure CHOOSEBADREGION recognizes that one of the idle persons Hübner, Müller, or Schmidt is required to serve an additional shift. To achieve this, CHOOSEBADREGION looks for days when one of these persons is surplus and stores the corresponding variables in a pool — this pool may also be filled by some other heuristics. Finally, CHOOSEBADREGION makes random choices from this pool and returns the resulting set of variables as a description of the next step of repair. In this example, taking the variable referring to Günther Hübner on Friday the 1st enables the system to compensate an additional early-morning shift of Günther Hübner on Sunday

the 17th. Often, only a very few number of opportunities is available to cope with such situations.

The lower roster in Fig. 9 shows the result of such repair steps that is available in about one minute. On larger problems, finding first solutions that satisfy the constraints on minimal crew attendance without exceeding the working time may take about five minutes. Running times of more than half an hour have been reported on problems with inappropriate assignments of working time models.

5 Conclusion

Based on experiences with the *ORBIS Dienstplan* system [12] — a nurse rostering system that is already used in many German hospitals — this paper describes how constraint processing can be used to implement automatic rostering systems that can be flexibly adopted to different situations at different hospitals. In practice, the current formalization of the rostering problem is still not complex enough to enable fully automatic planning procedures. Thus, complexity of the problem representation can be expected to grow in the future. Additionally, nurse rostering problems have many varying parameters: Working time accounts, demands on crew attendance, set of used shifts, working time models. Hence, rostering requires both, a flexible formalism for representing the currently known and future variants of the problem, and a robust search procedure that is able to cope with quite different problem instances and that allows the responsible managers to intervene into the planning process. This paper shows that constraint reasoning meets all these demands if the state of the art is extended in several directions.

The paper introduces a new notion of fuzzy or non-crisp constraints — constraints that may be partially violated and partially satisfied — in *hierarchical constraint optimization problems (HCOP)* to improve problem representation and problem solving. In contrast to standard formalisms of fuzzy constraints [17, 7] that sum up degrees of constraint violation taking the minimum or maximum, the formalism as introduced in this paper uses more flexible operations to combine different constraints. As a consequence, the well known branch&bound has been extended to detect conflicts with fuzzy constraints by constraint propagation. Of course, the proposed framework relates to other formalizations of soft constraints like for instance *valued constraint satisfaction (VCSP)* [3, 16], especially lexicographic VCSP. However, the framework of Section 2.1 uses fuzzy sets of constraints as a valuation structure that is idempotent but not totally ordered and this framework is able to borrow algorithms for the propagation of non-binary constraints from *constraint logic programming (CLP)*. VCSP neglects the advantages of specialized algorithms for constraint propagation [13]. Lexicographic VCSP can be translated into HCOPs with binary constraints of type *ext* that all have the same weight.

In combination with new extensions of generic search algorithms, this framework forms the basis of a nurse rostering system that differs in mainly two points from other constraint-based prototypes on this application [1, 22]:

– Fuzzy constraints allow the integration of very fine-grained optimization tasks. Such constraints have been used to optimize the amount of working time and the

presence on the ward. In contrast, traditional frameworks for constraint processing in nurse rostering consider only crisp constraints which are either completely violated or satisfied.
- The traditional constraint-based approach for solving optimization tasks is to use an extension of the branch&bound. Unfortunately, performance of tree search algorithms is very sensitive to even minor changes in the problem representation. For instance INTERDIP [1] tries to overcome this problem by use of highly developed heuristics on problem decomposition. Unfortunately, these heuristics rely on properties of the available working shifts which are, in practice, parameters of the problem that vary from hospital to hospital. In contrast, *ORBIS Dienstplan* integrates the branch&bound into search by iterative improvements. The branch&bound is used to enable the optimization of more than one variable assignment within one improvement step. The resulting search algorithm converges quickly on good rosters and, additionally, enables a more natural integration of user interactions.

Unfortunately, the ability to deal with complex improvement steps affecting more than one variable increases the number of applicable improvement steps compared to standard local search where only a single value assignment gets changed within each step of repair. Therefore, section 4 illustrates a heuristic that is very successful in finding the reason for deficiencies of the best yet found roster. Such heuristics are able to guide the proposed local search effectively.

References

1. Slim Abdennadher and Hans Schlenker. INTERDIP – an interactive constraint based nurse scheduler. In *PACLP-99: Proceedings of the International Conference on Practical Applications of Constraint Technology and Logic Programming*, Practical Applications Expo, London, 1999.
2. Fahiem Bacchus and Paul van Run. Dynamic variable ordering in CSPs. In Ugo Montanari and Francesca Rossi, editors, *CP–95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 258–275. Springer Verlag, 1995.
3. Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparisons. In M. Jampel, editor, *Over-Constrained Systems*. Springer Verlag, 1996.
4. Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5:233–270, 1992.
5. S. J. Darmoni, A. Fajner, A. Leforestier, and N. Mahè. Horoplan: Computer-assisted nurse scheduling using constraint-based programming. *Journal of the Society for Health Systems*, 5:41–54, 1995.
6. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the international conference on fifth generation computer systems*, 1988.
7. Didier Dubois, Hélène Fargier, and Henri Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. of the 2nd IEEE International Conference on Fuzzy Systems*, pages 1131–1136, San Francisco, CA, 1993.
8. François Fages, Julian Fowler, and Thierry Sola. Handling preferences in constraint logic programming with relational optimization. In M. Jampel, editor, *Over-Constrained Systems*. Springer Verlag, 1996.

9. Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
10. J. Gaschnig. A general backtrack algorithm that eliminates most redundant checks. In *Proceedings IJCAI-77*, Cambridge (MA), 1977.
11. Harald Meyer auf'm Hofe. Partial satisfaction of constraint hierarchies in reactive and interactive configuration. In Walter Hower and Zsófia Ruttkay, editors, *ECAI-96 Workshop on Non-Standard Constraint Processing*, pages 61–72, 1996.
12. Harald Meyer auf'm Hofe. ConPlan/ SIEDAplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *PACT-97: Proceedings of the Third International Conference on the Practical Application of Constraint Technology*, pages 257–272, Practical Application Expo, London, April 1997.
13. Harald Meyer auf'm Hofe. *Kombinatorische Optimierung mit Constraintverfahren — Problemlösung ohne anwendungsspezifische Suchstrategien*, volume 242 of *DISKI – Dissertationen zur Künstlichen Intelligenz*. Infix, akademische Verlagsgesellschaft, 2000. ISBN 3-89838-242-7.
14. Harald Meyer auf'm Hofe and Andreas Abecker. ConPlan: Solving scheduling problems represented by soft constraints. In *ISIAC-98: Proceedings of the Int' Symp' on Intelligent Automation and Control*, pages 245–250, Anchorage, Alaska, USA, 1998.
15. Nils Nilsson. *Principles of Artificial Intelligence*, chapter 3.2. Symbolic Computation. Springer, Berlin, 1982.
16. Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris Mellish, editor, *IJCAI-95. Proceedings, 14th International Joint Conference on Artificial Intelligence*, pages 631–637. Morgan Kaufmann, 1995.
17. Paul Snow and Eugene C. Freuder. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. In *Proc. of the 8^{th} biennal conf. of the canadian society for comput. studies of intelligence*, pages 227–230, May 1990.
18. Olaf Steinmann, Antje Strohmeier, and Thomas Stützle. Tabu search vs. random walk. In G. Brewka, C. Habel, and B. Nebel, editors, *KI-97: Advances in Artificial Intelligence*, volume 1303 of *LNAI*, pages 337–348. Springer Verlag, 1997.
19. E. Tsang. *Foundations of Constraint Satisfaction*. Computation in Cognitive Science. Academic Press, London, 1993. ISBN 0-12-701610-4.
20. Mark Wallace, Stefano Novello, and Joachim Schimpf. $\mathrm{Ecl}^{i}\mathrm{ps}^{e}$ — a platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, 1997.
21. Richard J. Wallace. Analysis of heuristic methods for partial constraint satisfaction problems. In *CP–96: Proceedings of the Second International Conference on Principles and Practice of Constraint Processing*, pages 482–496. Springer Verlag, 1996.
22. Georges Weil and Kamel Heus. Eliminating interchangeable values in the nurse scheduling problem formulated as a constraint satisfaction problem. In *CONSTRAINT–95: The FLAIRS–95 international workshop on constraint-based reasoning*, Melbourne Beach, FL, USA, April 1995.
23. Lofti A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.