Fundamenta Informaticae XX (2016) 1–24
DOI 10.3233/FI-2012-0000
IOS Press

Shift Design with Answer Set Programming *

Michael Abseher [C]
TU Wien, Austria

Nysret Musliu
TU Wien, Austria

Stefan Woltran
TU Wien, Austria

Martin Gebser
University of Potsdam, Germany

Torsten Schaub[†]
University of Potsdam, Germany
INRIA Rennes, France

Abstract. Answer Set Programming (ASP) is a powerful declarative programming paradigm that has been successfully applied to many different domains. Recently, ASP has also proved successful for hard optimization problems like course timetabling and travel allotment. In this paper, we approach another important task, namely, the shift design problem, aiming at an alignment of a minimum number of shifts in order to meet required numbers of employees (which typically vary for different time periods) in such a way that over- and understaffing is minimized. We provide an ASP encoding of the shift design problem, which, to the best of our knowledge, has not been addressed by ASP yet. Our experimental results demonstrate that ASP is capable of improving the best known solutions to some benchmark problems. Other instances remain challenging and make the shift design problem an interesting benchmark for ASP-based optimization methods.

1. Introduction

Answer Set Programming (ASP) [13] is a declarative formalism for solving hard computational problems. Thanks to the power of modern ASP technology [24], ASP was successfully used in various application areas, including product configuration [38], decision support for space shuttle flight controllers [33], team building and scheduling [36], and bio-informatics [26]. Recently, ASP also proved successful

*This work extends a preliminary workshop paper [2] presented at ASPOCP'15. A short version [3] appeared at LPNMR'15.
[C]Corresponding author
[†]Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

for optimization problems that had not been amenable to complete methods before, for instance in the domains of timetabling [8] and allotment [20].

In this paper, we investigate the application of ASP to another important domain, namely, workforce scheduling [16]. Finding appropriate staff schedules is of great relevance because work schedules influence health, social life, and motivation of employees at work. Furthermore, organizations in the commercial and public sector must meet their workforce requirements and ensure the quality of their services and operations. Such problems appear especially in situations where the required number of employees fluctuates throughout time periods, while operations dealing with critical tasks are performed around the clock. Examples include air traffic control, personnel working in emergency services, call centers, etc. In fact, the general employee scheduling problem includes several subtasks. Usually, in the first stage, the temporal requirements are determined based on tasks that need to be performed. Further, the total number of employees is determined and the shifts are designed. In the last phase, the shifts and/or days off are assigned to the employees. For shift design [32], employee requirements for a period of time, constraints about the possible starts and lengths of shifts, and limits for the average number of duties per week are considered. The aim is to generate solutions consisting of shifts (and the number of employees per shift) that fulfill all hard constraints, while minimizing the number of distinct shifts as well as over- and understaffing. This problem has been addressed by local search techniques, including a min-cost max-flow approach [32] and a hybrid method combining network flow with local search [17]. These techniques have been used to successfully solve randomly generated examples and problems arising in real-world applications.

Although the aforementioned state-of-the-art approaches for the shift design problem are able to provide optimal solutions in many cases, obtaining optimal solutions for large problems is still a challenging task. Indeed, for some instances the best solutions are still unknown. Therefore, the application of exact techniques like ASP is an important research target. More generally, it is interesting to see how far an elaboration-tolerant, general-purpose approach such as ASP can compete with dedicated methods when tackling industrial problems. Our ASP solution is based on the first author's master thesis [1] and relies on sophisticated modeling and solving techniques, whose application provides best practice examples for addressing similarly demanding use cases. On the one hand, we demonstrate how order encoding techniques [14] can be used in ASP for modeling complex interval constraints. On the other hand, our empirical evaluation contrasts traditional model-guided[1] optimization techniques with orthogonal core-guided techniques [30], revealing another case in which the latter have an edge over the former. While our experiments show that the shift design problem provides challenging benchmarks for state-of-the-art ASP technology, we are able to identify yet unknown global optima for some hard instances.

2. The Shift Design Problem

To begin with, let us introduce the shift design problem. Our problem formulation follows the one in [32], where an input specifies the following:

- consecutive *time slots* $[a_0, a_1), [a_1, a_2), \ldots, [a_{n-1}, a_n)$, all of them with the duration $slotlength$. We call this collection of time slots the *planning horizon* or, synonymously, the *planning period* of the problem. Often, this collection of time slots is partitioned into d days, where each of the

[1]That is, branch-and-bound based strategies; the term 'model-guided' was coined in [5].

shift type	$min\text{-}start$	$max\text{-}start$	$min\text{-}length$	$max\text{-}length$
M	07:00	08:00	07:00	09:00
D	10:30	11:30	07:00	09:00
A	14:00	16:00	07:00	08:00
N	22:00	24:00	07:00	09:00

Table 1. Possible shift types

days consists of the same number $daylength$ of time slots. Furthermore, each time slot $[a_i, a_{i+1})$ is associated with a number w_i of employees who should be present during the slot.

- *shift types* $\{t_1, \ldots, t_m\}$ with the associated parameters $min\text{-}start$ and $max\text{-}start$, representing the earliest and latest start, and $min\text{-}length$ and $max\text{-}length$, representing the minimum and maximum length of a shift. An example of such shift types is given in Table 1. For convenience, the discrete parameter values of the respective shift types are represented by the corresponding day times as this is the most likely input format one will receive from the managers of a company.

The aim is to generate a collection $s_1, \ldots, s_k$ of shifts. Each shift s_i is determined by its $start$ and $length$, which must belong to some shift type. Additionally, each s_i is associated with parameters $s_i.workers_j$ indicating the number of employees assigned to s_i on each day $j \in \{1, \ldots, d\}$ of the planning period. Note that we consider cyclic planning periods, where the successor of the last time slot is equal to the first time slot.

In analogy to [17], we investigate the optimization of the following criteria: sum of *shortages* of workers at each time slot during the planning period, sum of *excesses* of workers at each time slot during the planning period, and the *number of shifts*.[2] Traditionally, the objective function to minimize is a weighted sum of the three components (although this kind of aggregation is not mandatory with ASP).

Related work. Solving the shift design problem by local search has been thoroughly investigated in [32], where a tabu search approach and several neighborhood relations were proposed. The algorithms have been included in the scheduling system OPA (Operating Hours Assistant), which is used for solving real-world shift design problems in several companies. The NP-hardness of the shift design problem was shown in [17], where also an improved local search technique and a hybrid method combining a min-cost max-flow approach with local search was introduced. The inclusion of breaks into shift design was considered in [10, 11, 18, 42]. A detailed overview of previous work on shift design and break scheduling is given in [19].

A related problem is shift scheduling. To solve this problem, mainly exact approaches based on Integer Programming have been applied. For instance, following the set-covering formulation due to [15], several methods were proposed [7, 9, 35, 41]. Other approaches include large neighborhood search [34], tabu search [40], etc. The original shift design problem differs in several aspects from the shift scheduling problem. In our setting, shifts are constructed for the whole week and cyclicity is taken into account. Furthermore, we consider the minimization of the number of shifts, and both over- and understaffing are allowed.

[2] In [32], additionally, the average number of duties per week is considered.

Finally, there is also the area of workforce management, where the focus is on the allocation of employees of different qualifications to tasks requiring particular skills. Similar to shift design and scheduling problems, constraints concerning the workload have to be taken into account. In [36], a concrete problem from this domain has been tackled via ASP, and the resulting system is tailored to the specific needs of the seaport of Gioia Tauro. From the conceptual point of view, the main difference is that the problem encoded in [36] is a classical allocation problem with optimization towards work balance, while the problem we tackle aims at an optimal alignment of shifts.

3. Answer Set Programming

This section gives a brief introduction to the stable model semantics of logic programs, serving as the basis for encoding the shift design problem in ASP below. In particular, we consider logic programs with choice rules [37] and weak constraints [28]. A *rule* r in such a program is an expression of the form

$$h \leftarrow a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n \tag{1}$$

where $a_1, \ldots, a_n$ are *atoms* of the form $s(t_1, \ldots, t_k)$, in which s is a *predicate* symbol and $t_1, \ldots, t_k$ are *terms*, viz. constants, variables, or functions, and $\sim$ stands for *default negation*. The head h of r is either an atom a, a choice $\{a\}$, or the special symbol $\bot$. If h is an atom and $n = 0$, we call r a *fact*, a *choice rule* if h is $\{a\}$, and an *integrity constraint* if h is $\bot$; we skip $\leftarrow$ or $\bot$, respectively, when writing rules (1) with $n = 0$ and integrity constraints below. A *weak constraint* c is an expression of the form

$$w@p, t_1, \ldots, t_k \;\leftsquigarrow\; a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n \tag{2}$$

where $a_1, \ldots, a_n$ are atoms and $t_1, \ldots, t_k$ are terms. Moreover, the distinguished terms w and p provide the *weight* and *priority* of c. A *logic program* P is a set of rules and weak constraints. In the first-order case, terms occurring in P may include arithmetic expressions, and atoms may be based on relational operators like "$<$". On the other hand, a term, atom, rule, weak constraint, or program is *ground* if it does not include variables, arithmetic expressions, or relational operators. A first-order program P stands for the set $grd(P)$ of all instances of rules and weak constraints constructible by substituting ground terms for variables and evaluating arithmetic expressions as well as relational operators in the standard way. For details on ground instantiation, we refer the interested reader to [21, 25].

For example, the first-order program P_1 consisting of

$$\begin{array}{rcl}
p(1) & & \\
\{q(2)\} & & \\
r(X) & \leftarrow & p(X), X < 2, {\sim}r(X{+}1) \\
r(X) & \leftarrow & q(X), 1 < X, {\sim}r(X{-}1) \\
 & \leftarrow & r(X), q(Y), X < Y \\
X@1 & \leftsquigarrow & r(X) \\
1@X & \leftsquigarrow & X \in [1,2], {\sim}q(X)
\end{array}$$

yields a ground instantiation $grd(P_1)$ as follows:

$$\begin{array}{rcl}
p(1) & & \\
\{q(2)\} & & \\
r(1) & \leftarrow & p(1), \sim r(2) \\
r(2) & \leftarrow & q(2), \sim r(1) \\
 & \leftarrow & r(1), q(2) \\
1@1 & \leftsquigarrow & r(1) \\
2@1 & \leftsquigarrow & r(2) \\
1@1 & \leftsquigarrow & \sim q(1) \\
1@2 & \leftsquigarrow & \sim q(2)
\end{array}$$

Note that variables, denoted by uppercase letters in P_1, are substituted by constants 1 and 2 in $grd(P_1)$ such that relational operators, if any, in rules and weak constraints hold. Then, atoms based on relational operators are dropped and arithmetic expressions mapped to the result of their evaluation.

The semantics of a logic program P is given by its stable models, which are particular sets of (true) ground atoms as defined in the following. The *reduct* P^X relative to a set X of ground atoms is the set of all rules (1) and weak constraints (2) in $grd(P)$ such that $\{a_1, \dots, a_m\} \subseteq X$, $\{a_{m+1}, \dots, a_n\} \cap X = \emptyset$, and $a \in X$ if $h = \{a\}$ is a choice for a rule (1). Then, X is a *stable model* of P if it is $\subseteq$-minimal among the sets of ground atoms such that, for all rules (1) in P^X, $\{a_1, \dots, a_m\} \subseteq X$ implies $h \in X$ or $a \in X$ if $h = \{a\}$. Note that P^X cannot include integrity constraints relative to a stable model X, as the head $\bot$ is not an atom and thus $h \notin X$. Moreover, the $\subseteq$-minimality condition expresses that each atom $a \in X$ must be derivable from P^X, and thus a must occur in the head of at least one rule in P^X.

The stable models of P_1 above are formed as follows. First, the fact $p(1)$ belongs to the reduct relative to any set of ground atoms, so that each stable model must include $p(1)$. Since $\{q(2)\}$ is a choice, the atom $q(2)$ may be omitted, in which case rules depending positively on $q(2)$ are not contained in the reduct. Then, $r(2)$ is underivable, so that $p(1)$ and $\sim r(2)$ yield $r(1)$, leading to the stable model $X_1 = \{p(1), r(1)\}$ of P_1. On the other hand, if $q(2)$ is made true in view of the choice $\{q(2)\}$, the integrity constraint in $grd(P_1)$ asserts that $r(1)$ must be false, which in turn leads to $r(2)$ and the second stable model $X_2 = \{p(1), q(2), r(2)\}$ of P_1. Note that weak constraints do not affect these stable models.

Weak constraints allow for selecting optimal stable models of a logic program P. For a set X of ground atoms, let $\Sigma(P^X)$ be the set of all tuples $w@p, t_1, \dots, t_k$ from weak constraints (2) in P^X such that w and p are integers. Then, X is *dominated* by another set Y of ground atoms if there is an integer i such that $\sum_{(w@i,t_1,\dots,t_k)\in\Sigma(P^Y)} w < \sum_{(w@i,t_1,\dots,t_k)\in\Sigma(P^X)} w$ and $\sum_{(w@j,t_1,\dots,t_k)\in\Sigma(P^Y)} w = \sum_{(w@j,t_1,\dots,t_k)\in\Sigma(P^X)} w$ for all integers $j > i$. In turn, a stable model X of P is *optimal* if it is not dominated by any other stable model Y of P. Given this, the idea of ASP is to encode a computational problem by a first-order program such that its (optimal) stable models provide (preferred) solutions to arbitrary instances of the problem.

Reconsidering P_1, the weak constraints in $P_1^{X_1}$ give $\Sigma(P_1^{X_1}) = \{1@1, 1@2\}$ for $X_1 = \{p(1), r(1)\}$, while $P_1^{X_2}$ yields $\Sigma(P_1^{X_2}) = \{2@1, 1@1\}$ for $X_2 = \{p(1), q(2), r(2)\}$. Since $\sum_{(w@2)\in\Sigma(P_1^{X_2})} w = 0 < 1 = \sum_{(w@2)\in\Sigma(P_1^{X_1})} w$ for tuples of priority 2, X_1 is dominated by the optimal stable model X_2 of P_1, regardless of $\sum_{(w@1)\in\Sigma(P_1^{X_1})} w = 1 < 3 = \sum_{(w@1)\in\Sigma(P_1^{X_2})} w$ for tuples of the smaller priority 1.

4. Shift Design in ASP

To begin with, Section 4.1 lays some formal foundations for our ASP approach to shift design. This provides the basis for the fact format of problem instances specified in Section 4.2, as well as the correspondence between problem schedules and stable models of the ASP encoding developed in Section 4.3.

4.1. Formal Preliminaries

As mentioned in Section 2, an instance of the shift design problem is characterized by

- a number $daylength$ of time slots (of duration $slotlength$) per day along with a number d of days, determining the total number $n = d \times daylength$ of slots,
- desired numbers w_i of employees for the time slots indexed by $i \in \{0, \dots, n-1\}$,
- a collection $\{t_1, \dots, t_m\}$ of shift types, each associated with parameters $min\text{-}start$, $max\text{-}start$, $min\text{-}length$, and $max\text{-}length$, where $0 \le t.min\text{-}start \le t.max\text{-}start < daylength$ and $0 < t.min\text{-}length \le t.max\text{-}length \le n$ for $t \in \{t_1, \dots, t_m\}$. (The conditions express that the start times of shift types must correspond to times in a day and that lengths must not exceed the planning horizon, given that the planning period is cyclic and a shift cannot include any slot twice.)

The shift types $\{t_1, \dots, t_m\}$ induce a set

$$S = \left\{ (s.start, s.length) \;\middle|\; \begin{array}{r} t \in \{t_1, \dots, t_m\}, s.start \in \{t.min\text{-}start, \dots, t.max\text{-}start\}, \\ s.length \in \{t.min\text{-}length, \dots, t.max\text{-}length\} \end{array} \right\}$$

of admissible shifts $s \in S$, each characterized by its start slot within a day and its length. For example, the instance shown on the left in Figure 1 can be specified in terms of a $slotlength$ of 3 hours, a $daylength$ of 8 slots, and $d = 1$ for a planning horizon of one day. This yields the time slots indexed $0, \dots, 7$, whose associated numbers of employees are displayed in the grid at the bottom left, e.g., $w_0 = 1$ and $w_7 = 3$. Moreover, the admissible shifts, whose potential assignments (for day 1) are indicated above the grid, are induced, e.g., by shift types as follows:

shift type	$min\text{-}start$	$max\text{-}start$	$min\text{-}length$	$max\text{-}length$
t_1	06:00	06:00	06:00	12:00
t_2	12:00	21:00	06:00	12:00

These types represent the set $S = \{(2, l), (4, l), (5, l), (6, l), (7, l) \mid l \in \{2, 3, 4\}\}$ of shifts, whose start slots 2 and $4, \dots, 7$ stand for beginnings at day time 06:00, 12:00, 15:00, 18:00, or 21:00, respectively, while $l \in \{2, 3, 4\}$ expresses a duration of 6, 9, or 12 hours. Note that some $s \in S$, e.g., those with $s.start = 7$, wrap into the beginning of the planning period, reflecting the cyclic interpretation of schedules. However, if another day were added, i.e., for $d = 2$, the extended horizon would include eight additional slots, and shifts assigned on the second day (eight slots to the right) would wrap over.

A schedule for the given instance, utilizing the shifts $s_1 = (2, 4)$, $s_2 = (4, 4)$, and $s_3 = (7, 4)$ from S, is shown on the right in Figure 1. Note that several workers can be assigned to each shift, and the number of assigned workers can also vary from day to day. In view of the planning horizon of one

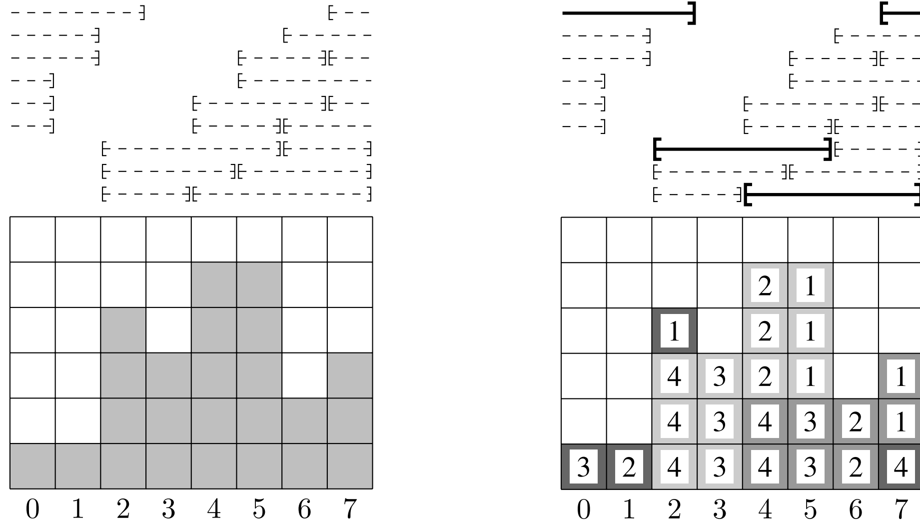

Figure 1. Work demands over a day with admissible shifts indicated above (left) and the unique optimal schedule with shifts of length 4 starting from slots 2, 4, and 7, assigned to three, two, or one employee, respectively (right)

day in our example, the displayed schedule is characterized by $s_1.\mathit{workers}_1 = 3$, $s_2.\mathit{workers}_1 = 2$, $s_3.\mathit{workers}_1 = 1$, and $s.\mathit{workers}_1 = 0$ for all $s \in S \setminus \{s_1, s_2, s_3\}$. The resulting coverage of slots by employees is indicated in the grid at the bottom right, where the numbers labeling blocks provide the *residual lengths* of assigned shifts at each slot. In fact, for each slot $i \in \{0, \dots, n-1\}$, the grid visualizes a multiset

$$L_i = \left[\big((k + s.\mathit{length}) - (i + c)\big)^{s.\mathit{workers}_j} \;\middle|\; \begin{array}{l} s \in S, j \in \{1, \dots, d\}, c \in \{0, n\}, \\ k = s.\mathit{start} + (j-1) \times \mathit{daylength}, \\ k \leq i + c < k + s.\mathit{length} \end{array} \right]$$

where $\big((k + s.\mathit{length}) - (i + c)\big)^{s.\mathit{workers}_j}$ stands for $s.\mathit{workers}_j$ repetitions of a residual length $k + s.\mathit{length} - (i + c)$, usually obtained by subtracting the index i from the end of s marked by $k + s.\mathit{length}$ for a day j. Moreover, adding $c = n$ to i captures overwrapping shifts including slots at the beginning of the planning period. E.g., considering $s_3 = (7, 4)$ and $s_3.\mathit{workers}_1 = 1$, we have $k = s_3.\mathit{start} = 7$ and $k + s.\mathit{length} = 11$, and in addition to a residual length 4 for $c = 0$ at slot 7, using $c = 1$ captures residual lengths of 3, 2, or 1, respectively, at slots 0, 1, and 2. Also note that $s_1.\mathit{workers}_1 = 3$ for $s_1 = (2, 4)$ maps to three occurrences of 4 in $L_2 = [4, 4, 4, 1]$, where the assignment of three workers to s_1 is reflected by $L_2 \setminus [l - 1 \mid l \in L_1] = [4, 4, 4, 1] \setminus [1] = [4, 4, 4]$. Unlike that, for $L_3 = [3, 3, 3]$, the difference $L_3 \setminus [l - 1 \mid l \in L_2] = [3, 3, 3] \setminus [3, 3, 3, 0] = []$ yields that no shift starts from slot 3. That is, comparing the multisets associated with neighboring slots provides a way to identify start slots, lengths, and assigned workers of shifts, and our ASP encoding in Section 4.3 makes use of such an alternative representation.

For a given schedule, the resulting number of employees at a slot $i \in \{0, \dots, n-1\}$ is simply the cardinality $|L_i|$ of the multiset of residual lengths associated with i. Given this, $\mathit{deviation}_i = |L_i| - w_i$ indicates over- or understaffing relative to the desired number w_i of employees in terms of positive or negative outcomes, respectively. It can be desirable to restrict either or both kinds of deviation via limits

excess and $\mathit{shortage}$ (using ∞ as value for expressing no limitation), and we say that a schedule is *legal*, if $\mathit{deviation}_i \leq \mathit{excess}$ and $-\mathit{deviation}_i \leq \mathit{shortage}$ for all $i \in \{0, \dots, n-1\}$, or illegal otherwise.

In practice, *optimization criteria* are important for distinguishing preferred schedules among the candidates that are legal. To this end, we map any schedule to three quality measures as follows:

$$\begin{aligned}\mathit{Deviation}^{+} &= \textstyle\sum_{i=0}^{n-1} \max\{0, \mathit{deviation}_i\} \\ \mathit{Deviation}^{-} &= \textstyle\sum_{i=0}^{n-1} \max\{0, -\mathit{deviation}_i\} \\ \mathit{Selected} &= |\{s \in S \mid \textstyle\sum_{j=1}^{d} s.\mathit{workers}_j > 0\}|\end{aligned}$$

That is, $\mathit{Deviation}^{+}$ accumulates overstaffing by summing up excesses of desired numbers of employees, $\mathit{Deviation}^{-}$ analogously captures understaffing, and $\mathit{Selected}$ provides the number of shifts to which employees are assigned on at least one day. To customize the accumulation of these measures m, we assume that each of them is associated with two non-negative integers specifying a priority $p(m)$ and a weight $w(m)$. When priorities $p_1, \dots, p_k$ such that $p_1 > \dots > p_k$ are used (where $k \leq 3$ in view of the three measures at hand), the quality of a schedule is expressed by a vector $\langle \sum_{p(m)=p_j} w(m) \times m \rangle_{j=1}^{k}$. Given this, a schedule is *optimal*, if its quality vector is lexicographically smallest among those of all legal schedules, so that priorities are handled hierarchically from greater to smaller ones. For instance, taking $p(\mathit{Deviation}^{-}) = 3$, $p(\mathit{Deviation}^{+}) = 2$, $p(\mathit{Selected}) = 1$, and $w(\mathit{Deviation}^{-}) = w(\mathit{Deviation}^{+}) = w(\mathit{Selected}) = 1$, the schedule on the right in Figure 1 yields $\langle 0, 0, 3 \rangle$, i.e., there is neither under- nor overstaffing, and employees are assigned to three shifts. One can further check that this schedule is optimal (w.r.t. the priorities and weights), while any other schedule without over- and understaffing must utilize at least four shifts.

Finally, note that we have not specified explicit restrictions on the number $s.\mathit{workers}_j$ of employees that can at most be assigned to a shift $s \in S$ for a day $j \in \{1, \dots, d\}$. (Recall that ∞ may be used for excess to impose no hard limitation.) However, the maximum number of desired employees over slots included in the shift provides a natural upper bound, as it is sufficient to avoid understaffing, while unnecessary overstaffing and employee assignments do never improve solution quality. More formally,

$$\max\{w_i \mid i \in \{0, \dots, n-1\}, c \in \{0, n\}, k = s.\mathit{start} + (j-1) \times \mathit{daylength}, k \leq i + c < k + s.\mathit{length}\}$$

gives a safe upper bound for $s.\mathit{workers}_j$. E.g., this maximum is $w_7 = 3$ for the shifts $(7, 2)$ and $(7, 3)$ of our example in Figure 1, since the desired numbers of employees for slots at the beginning of the planning period are $w_0 = w_1 = 1$. However, the shift $(7, 4)$ with greater length also includes slot 2, so that $w_2 = 4$ provides the maximum of employees possibly assigned to this shift. While the planning horizon in our example includes one day only, like desired numbers of employees, the upper bounds for shifts can vary from day to day, yet it remains trivial to read them off from a given problem instance.

4.2. Fact Format

The fact format of instances of the shift design problem follows the above elaborations. E.g., the facts describing the instance shown in Figure 1 are given in Figure 2. To begin with, facts of the form $\mathit{time}(S, T)$ provide the day time $T \in \{0, \dots, \mathit{daylength} - 1\}$ for each slot $S \in \{0, \dots, n-1\}$ of the planning period. Our example instance includes one day, divided into eight slots, corresponding to the times $0, \dots, 7$. Facts $\mathit{next}(S', S)$ specify consecutive slots, where S is usually $S' + 1$, except for the last slot whose successor is 0. For each slot S, a fact $\mathit{work}(S, N)$ gives the desired number $N = w_S$ of employees. Facts

$$\left\{\begin{array}{l}
\mathit{time}(0,0), \mathit{time}(1,1), \ldots, \mathit{time}(7,7), \mathit{next}(0,1), \mathit{next}(1,2), \ldots, \mathit{next}(7,0),\\
\mathit{work}(0,1), \mathit{work}(1,1), \mathit{work}(2,4), \mathit{work}(3,3), \mathit{work}(4,5), \mathit{work}(5,5),\\
\mathit{work}(6,2), \mathit{work}(7,3), \mathit{exceed}(1), \mathit{shorten}(1), \mathit{opt}(\mathit{shortage},3,1),\\
\mathit{opt}(\mathit{excess},2,1), \mathit{opt}(\mathit{select},1,1), \mathit{range}(2,2,1), \ldots, \mathit{range}(2,2,4),\\
\mathit{range}(2,3,1), \ldots, \mathit{range}(2,3,5), \mathit{range}(2,4,1), \ldots, \mathit{range}(2,4,5),\\
\mathit{range}(4,2,1), \ldots, \mathit{range}(4,2,5), \mathit{range}(4,3,1), \ldots, \mathit{range}(4,3,5),\\
\mathit{range}(4,4,1), \ldots, \mathit{range}(4,4,5), \mathit{range}(5,2,1), \ldots, \mathit{range}(5,2,5),\\
\mathit{range}(5,3,1), \ldots, \mathit{range}(5,3,5), \mathit{range}(5,4,1), \ldots, \mathit{range}(5,4,5),\\
\mathit{range}(6,2,1), \ldots, \mathit{range}(6,2,3), \mathit{range}(6,3,1), \ldots, \mathit{range}(6,3,3),\\
\mathit{range}(6,4,1), \ldots, \mathit{range}(6,4,3), \mathit{range}(7,2,1), \ldots, \mathit{range}(7,2,3),\\
\mathit{range}(7,3,1), \ldots, \mathit{range}(7,3,3), \mathit{range}(7,4,1), \ldots, \mathit{range}(7,4,4)
\end{array}\right\}$$

Figure 2. ASP facts providing the instance of the shift design problem shown on the left in Figure 1

$\mathit{exceed}(E)$ as well as $\mathit{shorten}(F)$ provide the deviation limits $E = \mathit{excess}$ and $F = \mathit{shortage}$ (or are omitted for value ∞ imposing no limitation). For instance, given $\mathit{excess} = \mathit{shortage} = 1$ and $w_7 = 3$, they express the upper bound 4 and the lower bound 2 for employees present at slot 7. Facts of the form $\mathit{range}(S, L, 1), \ldots, \mathit{range}(S, L, M)$ specify numbers of employees that can possibly be assigned to a shift $s = (T, L)$ for the day $j = \lceil \frac{S+1}{\mathit{daylength}} \rceil$, where T is the day time given by $\mathit{time}(S, T)$ and M is the maximum number of desired employees over slots included in the shift. That is, M provides the upper bound for $s.\mathit{workers}_j$ described at the end of Section 4.1. E.g., for shifts of length 2 or 3 starting from slot 7, the maximum is given by $w_7 = 3$, while the shift of length 4 also includes slot 2 with $w_2 = 4$.

Moreover, facts $\mathit{opt}(\mathit{shortage}, P, W)$, $\mathit{opt}(\mathit{excess}, P, W)$, and $\mathit{opt}(\mathit{select}, P, W)$ specify optimization criteria in terms of priority P and weight W. The values in Figure 2 represent $p(\mathit{Deviation}^-) = 3$, $p(\mathit{Deviation}^+) = 2$, $p(\mathit{Selected}) = 1$, and $w(\mathit{Deviation}^-) = w(\mathit{Deviation}^+) = w(\mathit{Selected}) = 1$. That is, the desired number of employees shall be present in the first place, then the amount of additional employees ought to be minimal, and third the number of utilized shifts should be as small as feasible.

4.3. Problem Encoding

Our ASP encoding of the shift design problem is shown in Figure 3. For a slot S, the intuitive reading of the predicate $\mathit{run}(S, L, I)$ is that at least I employees are assigned to shifts including S and $L - 1$ or more successor slots, i.e., $0 < I \leq |[l \in L_S \mid L \leq l]|$ holds for the multiset L_S of residual lengths at S. This is further refined by $\mathit{length}(S, L, I, J)$, telling that $0 < J \leq |[l \in L_S \mid l = L]|$ employees are assigned to shifts of exact residual length L, where $I + J \leq s.\mathit{workers}_j + 1$ (and $J \leq s.\mathit{workers}_j$) for the corresponding shift $s = (T, L)$ as given by $\mathit{time}(S, T)$ on day $j = \lceil \frac{S+1}{\mathit{daylength}} \rceil$. The predicate $\mathit{shift}(S, L, J)$ drops the latter condition that S is a potential start slot and simply expresses that $0 < J \leq |[l \in L_S \mid l = L]|$. Finally, $\mathit{start}(S, L, J)$ indicates that the J-th employee in a shift of exact residual length L is assigned to the corresponding shift with start slot S, i.e., $|[l \in L_{S'} \mid l = L + 1]| < J$ holds for the predecessor slot given by $\mathit{next}(S', S)$. A schedule is thus characterized by the number of (true) atoms of the form $\mathit{start}(S, L, J)$, yielding the employees assigned to a shift of length L

$$
\begin{array}{rcll}
\{run(S,L,I)\} & \leftarrow & range(S,L,I) & (3)\\
run(S,L,I) & \leftarrow & run(S',L{+}1,I), next(S',S), 0<L & (4)\\
run(S,L,I) & \leftarrow & run(S,L{+}1,I), 0<L & (5)\\
run(S,L,I{+}J) & \leftarrow & run(S,L{+}1,I), shift(S,L,J) & (6)\\
 & \leftarrow & run(S,L,I{+}1), 0<I, {\sim}run(S,L,I) & (7)\\
 & \leftarrow & work(S,N), exceed(E), run(S,1,N{+}E{+}1) & (8)\\
 & \leftarrow & work(S,N), shorten(F), F<N, {\sim}run(S,1,N{-}F) & (9)\\
length(S,L,I,1) & \leftarrow & range(S,L,I), run(S,L,I), {\sim}run(S,L{+}1,I) & (10)\\
length(S,L,I,J) & \leftarrow & length(S,L,I{+}1,J{-}1), 0<I, {\sim}run(S,L{+}1,I) & (11)\\
shift(S,L,J) & \leftarrow & length(S,L,I,J) & (12)\\
shift(S,L,J) & \leftarrow & shift(S',L{+}1,J), next(S',S), 0<L & (13)\\
start(S,L,J) & \leftarrow & range(S,L,J), next(S',S), shift(S,L,J), {\sim}shift(S',L{+}1,J) & (14)\\
W@P,S,I,shortage & \leftsquigarrow & opt(shortage,P,W), work(S,N), I\in[1,N], {\sim}run(S,1,I) & (15)\\
W@P,S,I,excess & \leftsquigarrow & opt(excess,P,W), work(S,N), run(S,1,I), N<I & (16)\\
W@P,T,L,select & \leftsquigarrow & opt(select,P,W), start(S,L,J), time(S,T) & (17)
\end{array}
$$

Figure 3. ASP encoding of the shift design problem

starting from slot S. For example, the schedule displayed in Figure 1 is described by a stable model containing $start(2,4,1)$, $start(2,4,2)$, $start(2,4,3)$, $start(4,4,1)$, $start(4,4,2)$, and $start(7,4,1)$. When additionally assigning one employee to the shift of length 2 starting from slot 6, this would be indicated by $start(6,2,3)$, as it adds to the two employees in a shift of residual length 2 starting from slot 4. However, the displayed schedule is the unique optimal solution, given that it matches the desired employees and uses a minimum number of shifts, viz. shifts of length 4 starting from slots 2, 4, and 7.

In more detail, the potential assignment of an I-th employee to a shift of length L starting from slot S is reflected by the choice rule (3) in Figure 3. Rule (4) propagates such an assignment to the $L-1$ successor slots of S included in the shift, where the residual length is successively decreased down to 1 in the last slot. For shifts with longer residual length L, rule (5) closes the interval between 1 and L, thus overturning any choice rules for potential starts of shifts of shorter length. Moreover, this allows for associating information about a J-th employee assigned to some shift of residual length L at slot S with a position $I+J$ (in the representation of L_S) whenever $I \leq |[l \in L_S \mid L < l]|$, as expressed by rule (6). The integrity constraint (7) asserts that the positions associated with assigned shifts must be ordered by residual length, thus guaranteeing a unique representation of the multiset L_S of residual lengths at S. This condition eliminates guesses on positions I reflecting the assignment of employees to a shift, and it also provides a shortcut making interconnections between positions of assigned shifts explicit, which in preliminary tests turned out as effective to improve search performance. The additional integrity constraints (8) and (9) are applicable whenever the deviation from a desired number of employees is bounded above or below, respectively. Note that it is sufficient to inspect atoms of the form $run(S,1,I)$ for appropriate positions I, given that residual lengths are propagated down to 1 via rule (5).

In order to derive the number of employees assigned to shifts of exact residual length L at slot S, where $s = (T, L)$ and $j = \lceil \frac{S+1}{daylength} \rceil$ are the corresponding shift and day, rule (10) marks positions $I \leq s.workers_j$ with 1 when the length L matches, i.e., $|[l \in L_S \mid L < l]| < I \leq |[l \in L_S \mid L \leq l]|$. Rule (11) then counts on the number of employees (backwards) over positions $|[l \in L_S \mid L < l]| + 1, \ldots, I - 1$. By projecting the positions I out, rule (12) then yields numbers $1, \ldots, J$ standing for employees possibly assigned to shift s on day j, where $J = |[l \in L_S \mid L \leq l]| - |[l \in L_S \mid L < l]| = |[l \in L_S \mid l = L]|$. In addition, employees assigned to longer shifts whose residual length decreases to L at S are propagated via rule (13). Finally, rule (14) captures numbers $|[l \in L_{S'} \mid l = L+1]| + 1, \ldots, |[l \in L_S \mid l = L]|$ expressing the difference between employees assigned to shifts of residual length L starting from S and those continued from the predecessor slot S' given by $next(S', S)$. As a consequence, a stable model represents a schedule in terms of sequences of the form $start(S, L, M), \ldots, start(S, L, N)$, telling that $N+1-M$ employees are assigned to a shift of length L starting from slot S.

It remains to assess the quality of a schedule, which is accomplished by means of the weak constraints (15), (16), and (17) for the three optimization criteria at hand. The penalty for deviating from a desired number of employees is characterized in terms of the priority P and weight W given in facts, a position I pointing to under- or overstaffing at a slot S, and a corresponding keyword $shortage$ or $excess$, respectively, for avoiding clashes with penalties due to the utilization of shifts. The latter include the keyword $select$ and map the start slot S of an assigned shift of length L to its day time T given by $time(S, T)$, so that the penalty $W@P$ is incurred at most once for a shift $s = (T, L)$, no matter how many and on how many days employees are actually assigned. The hierarchical treatment of penalties by priorities along with summation of weights within distinct term tuples t such that the condition c of some weak constraint $t \leftsquigarrow c$ holds then match the notion of optimality specified in Section 4.1. Since over- and understaffing at a slot are mutually exclusive for a schedule, the below variants of the weak constraints (15) and (16) can also be used for quality assessment, without modifying the optimal outcomes:

$$W@P, S, N+1-I, deviate \leftsquigarrow opt(shortage, P, W), work(S, N), I \in [1, N], {\sim}run(S, 1, I) \quad (15')$$

$$W@P, S, I-N, deviate \leftsquigarrow opt(excess, P, W), work(S, N), run(S, 1, I), N < I \quad (16')$$

These variants treat positive and negative deviations from a desired number of employees symmetrically by mapping them to similar term tuples, thus reducing the number of distinct tuples when the priority P and weight W in facts of the form $opt(shortage, P, W)$ and $opt(excess, P, W)$ coincide. In the next section, we empirically contrast both formulations and evaluate the impact on optimization performance.

A prevalent feature of our ASP encoding in Figure 3 is the use of closed intervals (starting from 1) to represent quantitative values such as residual lengths or numbers of assigned employees. The basic idea is similar to the so-called *order encoding* [14], which has been successfully applied to solve constraint satisfaction problems by means of SAT [39]. In our ASP encoding, rules (6), (10), and (11) take particular advantage of the order encoding approach by referring to one value, viz. $L+1$, for testing whether any shift with longer residual length than L is assigned. Likewise, the integrity constraints (8) and (9) as well as the weak constraints (15) and (16) (or (15') and (16')) focus on value 1, standing for any residual length, to determine the number of present employees. That is, the order encoding approach enables a compact formulation of existence tests and general conditions, which then propagate to all target values above or below a certain threshold.

5. Experiments

In the following, we present the experimental evaluation of our approach. The main part of the experiments is based on the solver clasp 3.1.3 [22]. Additional experiments, based on the solver WASP 2.0 [4], are provided in Section 5.4. All benchmark results were obtained using a machine with two Intel(R) Xeon(R) E5-2637 v3 @ 3.50GHz processors and 256GB RAM running Debian 8.2 (jessie). Each test run, using gringo 4.4.0 [25] for grounding and clasp 3.1.3 (or WASP 2.0 in case of Section 5.4) for solving, was bound to a single core and 8GB (64GB for WASP 2.0) RAM with a time limit of 60 minutes.

Preliminary tests showed best results with clasp's configuration `handy`. Additionally, we consider the `tweety` configuration, as it is clasp's default for ASP solving. More importantly, we compare the two main optimization strategies of clasp: (i) Branch-and-bound based optimization in hierarchical order of priorities (`--opt-strategy=bb,1`) and (ii) Unsatisfiable-core based optimization with disjoint-core preprocessing and implications to represent weak constraints (`--opt-strategy=usc,3`). With strategy (i), we utilize domain heuristics (`--dom-mod=4,8`), which in preliminary tests turned out to greatly accelerate the process of convergence towards an optimum. The heuristic parameters express that choices on atoms occurring in weak constraints shall always pick the truth value that falsifies the condition of a respective weak constraint; note that this modification is void for strategy (ii) because such truth values are pre-assigned anyway in the search for unsatisfiable cores. In case of balanced optimization criteria with common priority and weight, we also provide results for the encoding variant using the weak constraints (15') and (16') for a symmetric representation of shortage and excess deviations; this alternative encoding, activated through gringo's constant replacement, is indicated by `obj=1` below.

The traditional branch-and-bound strategy constitutes a *model-guided* approach that aims at successively producing solutions of descending costs until an optimum is found (by establishing the unsatisfiability of the problem with an even lower cost). In addition, the hierarchical variant of clasp [23] allows for non-uniform descents during optimization. For instance in multi-criteria optimization, this enables the consideration of criteria in the order of priority, rather than producing spurious (intermediate) solutions.

Core-guided approaches originated in the area of MaxSAT [12]. They rely on identifying and successively relaxing unsatisfiable subsets of weak constraints until a solution that is guaranteed to be optimal is obtained (see [30]). The implementation in clasp utilizes the core-guided optimization algorithm *oll* [6]. Its (optional) combination with disjoint-core preprocessing [29], as a side effect, provides a quick approximation of an optimum, while no intermediate solutions are obtained otherwise.

5.1. Problem Instances

We use instances of the shift design problem from four different benchmark sets. Below we briefly explain their basic structure and relevant characteristics. All benchmark sets are publicly available under the address `http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html`.[3] The data sets were first described in [31, 32] and also used in [17] for evaluating hybrid solving approaches.

DataSet1: The first data set contains 30 instances that can be solved without any deviation, since they were generated by first constructing a feasible assignment of employees to selected shifts (also called the *seed solution*), and then the resulting coverage values were taken as requirements in respective instances.

DataSet2: The second data set also consists of 30 instances, which are quite similar to those of the first data set, yet constructed with the intention to study the impact of the number of shifts in the best

[3]The ASP facts and encoding are provided at: `http://www.dbai.tuwien.ac.at/proj/Rota/DataSetASP.zip`

known solution on computation time. To this end, seed solutions were picked in such a way that the instances 1–10 should need at least 12 shifts to be solved exactly. The instances 11–20 are based on seed solutions of 16 shifts, and the remaining ten instances were constructed with seed solutions using 20 shifts. Di Gaspero et al. [17] note that their heuristic solving method was also able to find better solutions for some of the problem instances. In our experimental evaluation, we thus use their results for reference.

DataSet3: Di Gaspero et al. [17] highlight that in cases where an exact solution exists, the behavior of heuristics could be biased in comparison to the general case that there is no solution without deviation. To evaluate the solving efficiency on instances that cannot be solved exactly, the third data set contains 30 instances that were constructed in the same way as the two previous data sets, but this time, invalid shifts were added during the construction process. Such invalid shifts are not admitted in a solution, so that it is unlikely that an instance of the third data set can be solved without deviation. The instances 1–10 were constructed with seed solutions of 12 shifts (valid and invalid ones), and also the remaining instances are generated using the same scheme concerning the number of shifts as the second data set.

DataSet4: The fourth data set contains three problem instances, among which the first one is a complex real-world example to complement randomly generated instances. The second instance is almost identical to the fifth one in DataSet3, but the duration of time slots is halved. In this way, the second instance allows for investigating the impact of increasing the scheduling granularity. A similar approach is used for the third instance, but here the requirements are doubled instead of the number of time slots. Note that no best known fitness values have been published for the fourth data set in the paper by Di Gaspero et al. [17], which we use for our comparison here.

Shift design problems arising in practical contexts are in many cases similar to the instances considered here. This is due to the fact that, although the majority of the instances we investigate is randomly generated, the parameters (maximum number of workers per time slot in the seed solution, number of time slots, and the configuration of shift types) are based on real-world instances like the one in DataSet4.

5.2. Balanced Optimization Criteria

Tables 2–4 provide fitness values and runtimes for all data sets, taking a common priority and weight to penalize deviations as well as the utilization of shifts. In the second column of the tables, the best known fitness values are listed, and the columns to the right of it show the results we obtained using ASP.

The values given in the tables represent the median of five test runs per instance at the time of their termination. The reference values in columns for the best known fitness, taken from [17], are means over ten, for DataSet1 and DataSet2, or hundred, in case of DataSet3, trials with incomplete methods. An entry "$>$ 1h" denotes that the corresponding instance has not been solved within the time limit of one hour, and a dash in a column for the fitness expresses that clasp did not produce any solution within one hour. Given that our experiments are conducted without hard limits on the maximum shortage and excess and since we also do not restrict the maximum number of utilized shifts, we can directly compare our results to previous work on the same instances, and we will provide formerly unknown global optima for four instances. In fact, the fitness according to [17] is based on a balanced sum penalizing deviations and the utilization of shifts equitably, corresponding to weak constraints of common priority and weight.

In Table 2, we see that the branch-and-bound based optimization strategy (shown in columns headed by `--opt-strategy=bb,1`) is outperformed by the unsatisfiable-core based strategy (in columns headed by `--opt-strategy=usc,3`). Not surprisingly, branch-and-bound based optimization leads

Instance	Best Fitness [17]	`--opt-strategy=bb,1` `--dom-mod=4,8`						`--opt-strategy=usc,3`					
		`tweety`		`handy`		`handy, obj=1`		`tweety`		`handy`		`handy, obj=1`	
		Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	480	2820	> 1h	3780	> 1h	2760	> 1h	480	8.3	480	14.6	480	15.8
2	300	3000	> 1h	6000	> 1h	4470	> 1h	300	101.7	300	91.8	300	96.5
3	600	5160	> 1h	5940	> 1h	6480	> 1h	600	14.5	600	24.1	600	29.6
4	450	11730	> 1h	11310	> 1h	8640	> 1h	450	708.2	450	221.0	450	209.0
5	480	480	1001.7	2880	> 1h	2700	> 1h	480	6.3	480	7.6	480	7.8
6	420	420	167.2	420	1046.3	420	1053.1	420	2.8	420	4.0	420	4.1
7	270	5100	> 1h	5400	> 1h	2130	> 1h	270	110.9	270	115.7	270	108.0
8	150	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
9	150	10155	> 1h	8010	> 1h	5160	> 1h	150	1682.4	150	1982.0	150	1820.6
10	330	9390	> 1h	7800	> 1h	4920	> 1h	330	129.9	330	132.0	330	140.9
11	30	30	744.0	390	> 1h	30	674.7	30	212.2	30	200.7	30	230.4
12	90	5385	> 1h	5175	> 1h	4185	> 1h	90	934.8	90	846.0	90	928.6
13	105	10080	> 1h	7665	> 1h	4605	> 1h	105	1728.4	105	1449.8	105	1606.1
14	195	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
15	180	180	4.2	180	11.8	180	2.1	180	0.5	180	0.7	180	0.8
16	225	18090	> 1h	19275	> 1h	7980	> 1h	225	3505.6	225	3286.7	225	3451.7
17	540	11040	> 1h	9840	> 1h	9270	> 1h	540	333.6	540	302.4	540	659.4
18	720	7260	> 1h	8640	> 1h	6720	> 1h	720	13.4	720	23.5	720	22.1
19	180	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
20	540	1320	> 1h	2640	> 1h	—	> 1h	540	7.3	540	10.5	540	10.3
21	120	5640	> 1h	5430	> 1h	3735	> 1h	120	1138.6	120	1032.5	120	978.4
22	75	1500	> 1h	1455	> 1h	2415	> 1h	75	547.4	75	588.6	75	548.1
23	150	15660	> 1h	13245	> 1h	4965	> 1h	150	2363.0	150	2106.3	150	2265.1
24	480	480	690.7	900	> 1h	480	3497.0	480	4.9	480	5.9	480	6.3
25	480	11520	> 1h	11190	> 1h	5010	> 1h	480	422.0	480	323.3	480	369.1
26	600	1740	> 1h	3540	> 1h	1260	> 1h	600	7.8	600	12.7	600	14.1
27	480	6840	> 1h	5220	> 1h	5580	> 1h	480	14.2	480	21.0	480	22.4
28	270	2970	> 1h	2670	> 1h	3240	> 1h	270	29.0	270	37.4	270	40.1
29	360	10170	> 1h	9000	> 1h	3630	> 1h	360	150.5	360	138.7	360	259.3
30	75	1215	> 1h	2385	> 1h	765	> 1h	75	375.0	75	346.6	75	394.2

Table 2. Fitness values and runtimes for DataSet1

to a vast number of intermediate, non-optimal solutions, so that fitness values are improved rather slowly. In contrast, the unsatisfiable-core based approach takes more time to come up with intermediate solutions, but it produces much fewer of them before converging to an optimum. Another important point to mention is the fact that using domain heuristics on top of the branch-and-bound approach is crucial to improve the quality of obtained solutions and absolutely recommended when applying branch-and-bound based optimization to our ASP encoding.

Beyond that `--opt-strategy=usc,3` delivers global optima for all but three instances of the first data set, another interesting observation is that the performance of different configurations varies depending on the instance and the optimization strategy used. E.g., consider Instances 3 and 4 in Table 2. The `tweety` configuration turns out to be better than `handy` for Instance 3, no matter whether the encoding variant denoted by `obj=1` is used with the latter. On the other hand, for Instance 4, the `tweety`

Instance	Best Fitness [17]	`--opt-strategy=bb,1` `--dom-mod=4,8`						`--opt-strategy=usc,3`					
		`tweety`		`handy`		`handy, obj=1`		`tweety`		`handy`		`handy, obj=1`	
		Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	720	3120	> 1h	4020	> 1h	3720	> 1h	720	5.3	720	10.0	720	9.6
2	720	4680	> 1h	6180	> 1h	4320	> 1h	720	10.9	720	18.8	720	20.0
3	360	13320	> 1h	11760	> 1h	7470	> 1h	360	276.01	360	254.7	360	256.1
4	360	5880	> 1h	6540	> 1h	4860	> 1h	360	88.9	360	98.1	360	98.1
5	720	5280	> 1h	5340	> 1h	5340	> 1h	720	9.5	720	14.3	720	11.8
6	360	14160	> 1h	13050	> 1h	3300	> 1h	360	217.3	360	248.9	360	238.0
7	720	9180	> 1h	8460	> 1h	7380	> 1h	720	17.9	720	20.9	720	21.4
8	180	10635	> 1h	9480	> 1h	6270	> 1h	180	2056.8	180	1968.3	180	2038.7
9	360	12780	> 1h	9570	> 1h	7350	> 1h	360	177.7	360	193.0	360	197.0
10	660	5760	> 1h	7560	> 1h	5760	> 1h	660	15.0	660	24.4	660	31.9
11	480	9750	> 1h	8250	> 1h	4230	> 1h	480	1172.2	480	1239.4	480	1209.2
12	900	5460	> 1h	7320	> 1h	2700	> 1h	900	35.4	900	63.2	900	49.4
13	900	10620	> 1h	9900	> 1h	7260	> 1h	900	48.4	900	41.5	900	88.1
14	840	8040	> 1h	9420	> 1h	6480	> 1h	840	20.0	840	25.1	840	39.2
15	480	13620	> 1h	11070	> 1h	2190	> 1h	480	1103.6	480	798.8	480	1947.4
16	240	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
17	960	7920	> 1h	7920	> 1h	4260	> 1h	960	23.7	960	21.0	960	26.3
18	840	8760	> 1h	10560	> 1h	5520	> 1h	840	42.3	840	57.9	840	144.5
19	240	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
20	960	8280	> 1h	10140	> 1h	5940	> 1h	960	26.7	960	26.3	960	27.0
21	600	14940	> 1h	15030	> 1h	11610	> 1h	600	640.6	600	1159.7	600	1352.3
22	1080	11220	> 1h	11640	> 1h	9960	> 1h	1080	716.3	1080	176.8	1080	1316.9
23	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
24	600	12780	> 1h	11850	> 1h	9180	> 1h	600	1359.0	600	760.2	600	1056.1
25	600	16410	> 1h	15060	> 1h	9660	> 1h	600	2688.4	600	1047.9	600	1665.9
26	1020	11700	> 1h	7800	> 1h	6120	> 1h	1020	49.6	1020	60.4	1020	91.5
27	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
28	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
29	1140	10020	> 1h	11160	> 1h	7860	> 1h	1140	223.2	1140	89.2	1140	647.2
30	1020	13260	> 1h	9960	> 1h	2280	> 1h	1020	2600.3	1020	3316.3	3900	> 1h

Table 3. Fitness values and runtimes for DataSet2

configuration leads to a three times longer run with `--opt-strategy=usc,3` and deteriorates the fitness value with `--opt-strategy=bb,1`. That is, the average performance is primarily governed by the applied optimization strategy, while the effect of search parameter configurations is non-uniform (although clasp in branch-and-bound mode completes more instances with the `tweety` configuration). Moreover, the interplay between a strategy and the encoding of objectives can be crucial, as witnessed by `handy` with `obj=1`, whose symmetric representation of deviations improves branch-and-bound based optimization in comparison to the plain `handy` configuration for the vast majority of instances. Unlike that, unsatisfiable-core based optimization turns out to be agnostic here, and `obj=1` makes little difference relative to the plain `handy` and `tweety` configurations with `--opt-strategy=usc,3`. Given the similar nature of the second data set, the performance results in Table 3 yield analogous behavior.

While the previous outcomes already show that `--opt-strategy=usc,3` is a viable choice for

Instance	Best Fitness [17]	`--opt-strategy=bb,1` `--dom-mod=4,8`						`--opt-strategy=usc,3`					
		`tweety`		`handy`		`handy, obj=1`		`tweety`		`handy`		`handy, obj=1`	
		Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
DataSet3													
1	2386.80	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
2	7672.59	21330	> 1h	18780	> 1h	17130	> 1h	12930	> 1h	12840	> 1h	14730	> 1h
3	9582.14	20640	> 1h	18780	> 1h	15510	> 1h	10470	> 1h	**9540**	**2080.3**	**9540**	**3402.6**
4	6634.40	18480	> 1h	17070	> 1h	12630	> 1h	8700	> 1h	**6540**	**1728.7**	9690	> 1h
5	9996.00	16800	> 1h	16140	> 1h	14940	> 1h	13500	> 1h	13380	> 1h	14040	> 1h
6	2076.75	9135	> 1h	9345	> 1h	7275	> 1h	5445	> 1h	6300	> 1h	4485	> 1h
7	6075.00	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
8	8860.50	19320	> 1h	19440	> 1h	17370	> 1h	12660	> 1h	12870	> 1h	15720	> 1h
9	6036.90	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
10	2968.95	11130	> 1h	10140	> 1h	8010	> 1h	5940	> 1h	6810	> 1h	6540	> 1h
11	5490.90	21570	> 1h	20760	> 1h	16320	> 1h	9840	> 1h	10200	> 1h	8850	> 1h
12	4171.20	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
13	4662.00	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
14	9616.55	14820	> 1h	15480	> 1h	13560	> 1h	12900	> 1h	12720	> 1h	14580	> 1h
15	11445.00	24060	> 1h	24780	> 1h	23820	> 1h	13530	> 1h	13650	> 1h	16770	> 1h
16	10734.00	20460	> 1h	18120	> 1h	14100	> 1h	13800	> 1h	13560	> 1h	14880	> 1h
17	4729.05	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
18	6692.40	16650	> 1h	14280	> 1h	14580	> 1h	10440	> 1h	10950	> 1h	10680	> 1h
19	5157.45	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
20	9153.90	28110	> 1h	24420	> 1h	21030	> 1h	15540	> 1h	16920	> 1h	18060	> 1h
21	6053.55	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
22	12870.30	29640	> 1h	27840	> 1h	24450	> 1h	15750	> 1h	15600	> 1h	17640	> 1h
23	8384.24	19560	> 1h	18840	> 1h	15900	> 1h	13320	> 1h	12660	> 1h	20460	> 1h
24	10417.80	23460	> 1h	22980	> 1h	18900	> 1h	13800	> 1h	15720	> 1h	19680	> 1h
25	13204.80	23520	> 1h	19740	> 1h	19500	> 1h	**13020**	**43.3**	**13020**	**45.1**	**13020**	**111.14**
26	13117.80	37680	> 1h	33420	> 1h	28050	> 1h	22020	> 1h	20610	> 1h	21900	> 1h
27	10081.20	19020	> 1h	19200	> 1h	17760	> 1h	**10020**	**124.5**	**10020**	**1348.8**	**10020**	**207.2**
28	10603.80	20160	> 1h	20040	> 1h	18600	> 1h	14580	> 1h	13800	> 1h	17760	> 1h
29	6690.00	21870	> 1h	22290	> 1h	18000	> 1h	11010	> 1h	10290	> 1h	9120	> 1h
30	13723.80	24480	> 1h	23400	> 1h	19980	> 1h	17460	> 1h	17520	> 1h	20160	> 1h
DataSet4													
1	N/A	51600	> 1h	50040	> 1h	48360	> 1h	57600	> 1h	60060	> 1h	50460	> 1h
2	N/A	16860	> 1h	19680	> 1h	17700	> 1h	13260	> 1h	13170	> 1h	16560	> 1h
3	N/A	36360	> 1h	35280	> 1h	29100	> 1h	25980	> 1h	28560	> 1h	30180	> 1h

Table 4. Fitness values and runtimes for DataSet3 and DataSet4

tackling the shift design problem, in Table 4, presenting our results for the third and fourth data set, we see that our approach also works quite well on instances having no solutions without deviation from the requirements. In particular, we would like to draw the reader's attention to Instances 3, 4, 25, and 27 of the third data set. For these four instances, our approach allows us to find formerly unknown global optima (highlighted in boldface), thus improving on results obtained with incomplete methods [17]. In particular, the plain `handy` configuration, not switching the encoding as done with `obj=1`, turns out to

be robust by solving all of the four instances within the time limit. We note that the known solutions for other instances have been further improved recently [27], but to the best of our knowledge, the optimality of some solutions for instances of the third data set has been shown for the first time by our approach. This highlights the prospects of ASP-based optimization methods for shift design, for which instances of the third and fourth data set contribute challenging benchmarks in turn.

5.3. Hierarchical Optimization Criteria

In practical situations, deviations from requirements and the utilization of shifts may be more or less significant, so that the flexibility to customize priorities and weights of optimization criteria is of interest. To assess the capabilities of our approach in such application scenarios, we conducted experiments with hierarchical criteria, minimizing understaffing, overstaffing, and the number of shifts in decreasing order of priority. For computing optimal solutions in this setting, it is sufficient to provide $opt(shortage, 3, 1)$, $opt(excess, 2, 1)$, and $opt(select, 1, 1)$ via facts in the input, as illustrated in Figure 2. In view of the existence of exact solutions for instances of DataSet1 and DataSet2, we in the following concentrate on DataSet3 and DataSet4, where deviations and shift utilizations both impose non-trivial challenges.

Table 5 provides objective values and runtimes obtained with branch-and-bound and unsatisfiable-core based optimization, using the `handy` configuration of clasp. Note that the encoding variant denoted by `obj=1` above does not make any difference when shortage and excess are distinguished by priority, so that it does not contribute an additional alternative here. To the best of our knowledge, the reported objective values for shortage, excess, and shifts (the measures $Deviation^{-}$, $Deviation^{+}$, and $Selected$ described in Section 4.1) constitute new best known results w.r.t. the hierarchical criteria. Rows in gray highlight that the respective results in boldface supply global optima.

In contrast to Table 4, we observe that clasp provides us with global optima for far more instances. In fact, we are able to determine global optima for two out of three very large instances of DataSet4. That is, hierarchical criteria facilitate optimization because different and partially conflicting objectives can be tackled in separation. Nevertheless, the instances of DataSet3 for which global optima are obtained do not subsume those for which the same has been accomplished under balanced criteria, as reported in Table 4. This applies to Instances 4 and 27, while the promising results for unsatisfiable-core based optimization in Table 5 indicate that global optimization may be feasible for them when given more time. Although branch-and-bound based optimization remains clearly behind again, we note that its intermediate solution for Instance 27 is of better quality than the one obtained with unsatisfiable-core based optimization. This reminds of that branch-and-bound based optimization stays worthwhile as an anytime approach whenever global optimization via the unsatisfiable-core based strategy is beyond reach. Moreover, as demonstrated in [22], parallel portfolios, combining complementary optimization strategies as well as search parameter configurations, may significantly improve the overall robustness, even though such setups are beyond the scope of this paper in view of the vast number of options to compose portfolios.

5.4. Experiments with Other Solvers

In addition to our main evaluation based on clasp, we compared the solver WASP 2.0 [4], again using gringo 4.4.0 for grounding. The comparison is based on balanced optimization criteria as in Section 5.2 and takes the following configurations of WASP into account: `default` (no parameters used, core-

Instance	--opt-strategy=bb,1 --dom-mod=4,8 handy				--opt-strategy=usc,3 handy			
	Shortage	Excess	Shifts	Time	Shortage	Excess	Shifts	Time
DataSet3								
1	—	—	—	> 1h	—	—	—	> 1h
2	0	407	41	> 1h	**0**	**325**	**18**	**232.3**
3	0	729	39	> 1h	**0**	**673**	**15**	**116.3**
4	0	474	50	> 1h	0	415	51	> 1h
5	0	194	22	> 1h	**0**	**194**	**9**	**34.8**
6	0	353	55	> 1h	0	226	44	> 1h
7	—	—	—	> 1h	—	—	—	> 1h
8	0	485	42	> 1h	**0**	**367**	**15**	**386.6**
9	—	—	—	> 1h	—	—	—	> 1h
10	0	205	31	> 1h	**0**	**137**	**22**	**236.6**
11	0	803	55	> 1h	0	778	67	> 1h
12	—	—	—	> 1h	—	—	—	> 1h
13	—	—	—	> 1h	—	—	—	> 1h
14	0	229	19	> 1h	**0**	**215**	**13**	**324.5**
15	0	820	53	> 1h	**0**	**706**	**15**	**1083.8**
16	0	234	25	> 1h	**0**	**234**	**14**	**24.2**
17	—	—	—	> 1h	—	—	—	> 1h
18	0	377	48	> 1h	**0**	**265**	**21**	**747.0**
19	—	—	—	> 1h	—	—	—	> 1h
20	0	1081	58	> 1h	0	998	66	> 1h
21	—	—	—	> 1h	—	—	—	> 1h
22	0	834	66	> 1h	**0**	**661**	**16**	**1407.4**
23	0	291	29	> 1h	**0**	**232**	**17**	**341.3**
24	0	301	23	> 1h	**0**	**275**	**15**	**80.6**
25	0	311	25	> 1h	**0**	**266**	**17**	**218.5**
26	0	1066	60	> 1h	0	1049	84	> 1h
27	0	380	21	> 1h	0	393	26	> 1h
28	0	283	30	> 1h	0	229	18	> 1h
29	0	653	53	> 1h	0	520	64	> 1h
30	0	333	22	> 1h	**0**	**278**	**17**	**43.5**
DataSet4								
1	0	1736	8	> 1h	**0**	**1729**	**7**	**566.0**
2	0	513	38	> 1h	0	470	56	> 1h
3	0	430	23	> 1h	**0**	**388**	**9**	**2766.3**

Table 5. Objective values and runtimes for DataSet3 and DataSet4 w.r.t. hierarchical optimization criteria

guided), `basic` (model-guided), `opt` (model-guided), and `oll` (core-guided). With the `oll` configuration, we used the additional parameters `--enable-disjcores` and `--minimize-unsatcore`, as they showed best results in preliminary tests. In view of the sometimes higher memory requirements of WASP, we allowed 64GB RAM, again with a time limit of 60 minutes per run.

Table 6 shows the results of experiments on DataSet1. The first four columns provide instance IDs, best fitness values from [17], and the performance of clasp using `--opt-strategy=usc,3` along

Instance	Best	clasp		WASP							
	Fitness	`handy, usc3`		`default`		`basic`		`opt`		`oll`	
	[17]	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	480	480	14.6	480	13.0	45960	> 1h	480	> 1h	480	15.2
2	300	300	91.8	300	88.0	48870	> 1h	450	> 1h	300	86.3
3	600	600	24.1	600	19.8	68760	> 1h	600	> 1h	600	22.9
4	450	450	221.0	450	217.5	119040	> 1h	—	> 1h	450	205.3
5	480	480	7.6	480	7.1	27180	> 1h	480	946.8	480	6.9
6	420	420	4.0	420	6.2	21540	> 1h	420	261.9	420	5.5
7	270	270	115.7	270	111.1	59730	> 1h	270	> 1h	270	114.0
8	150	—	> 1h	—	> 1h	93345	> 1h	—	> 1h	150	3421.6
9	150	150	1982.0	150	1779.6	67485	> 1h	150	> 1h	150	1567.8
10	330	330	132.0	330	130.7	89400	> 1h	540	> 1h	330	136.4
11	30	30	200.7	30	202.5	22620	> 1h	30	264.1	30	195.3
12	90	90	846.0	90	819.1	44130	> 1h	90	> 1h	90	753.8
13	105	105	1449.8	105	1546.3	56640	> 1h	225	> 1h	105	1380.3
14	195	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
15	180	180	0.7	180	1.1	10560	> 1h	180	1.3	180	1.1
16	225	225	3286.7	—	> 1h	107820	> 1h	—	> 1h	225	3025.5
17	540	540	302.4	540	442.5	120270	> 1h	—	> 1h	540	253.1
18	720	720	23.5	720	22.5	68040	> 1h	2820	> 1h	720	17.1
19	180	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
20	540	540	10.5	540	8.9	31560	> 1h	540	> 1h	540	9.3
21	120	120	1032.5	120	1011.5	43650	> 1h	—	> 1h	120	882.4
22	75	75	588.6	75	670.1	36705	> 1h	75	2842.9	75	445.3
23	150	150	2106.3	150	2204.4	79185	> 1h	225	> 1h	150	1969.7
24	480	480	5.9	480	6.0	25500	> 1h	480	> 1h	480	5.4
25	480	480	323.3	480	517.8	105090	> 1h	—	> 1h	480	362.0
26	600	600	12.7	600	18.5	41520	> 1h	600	> 1h	600	12.5
27	480	480	21.0	480	16.8	66300	> 1h	480	> 1h	480	16.8
28	270	270	37.4	270	31.7	37710	> 1h	270	> 1h	270	34.0
29	360	360	138.7	360	135.1	70650	> 1h	2040	> 1h	360	131.5
30	75	75	346.6	75	353.0	29370	> 1h	75	1709.2	75	294.1

Table 6. Fitness values and runtimes for DataSet1 (WASP)

with the `handy` configuration, which in Section 5.2 turned out to be robust, for reference. The remaining columns give results obtained with WASP, where the core-guided `default` and `oll` approaches behave very similar to clasp. We observe that WASP with the `oll` approach is the only configuration that solves Instance 8 within the time limit, and otherwise it is often slightly faster than clasp. Interestingly, the model-guided `opt` approach also turns out to be competitive. In many cases, an optimum is found and only the final unsatisfiability check exceeds the time limit. However, similar to clasp's branch-and-bound based optimization, WASP's `basic` approach performs poorly in terms of resulting fitness values, while it produces the most (non-optimal) intermediate solutions.

Table 7 provides performance results on DataSet2. While WASP's `oll` approach was slightly more efficient than clasp on the "easy" instances of DataSet1, both solvers perform evenly here. In fact, they complete the same number of instances, and the `default` and `opt` approaches are competitive

Instance	Best Fitness [17]	clasp handy, usc3		WASP default		WASP basic		WASP opt		WASP oll	
		Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	720	720	10.0	720	8.4	42900	> 1h	720	> 1h	720	8.7
2	720	720	18.8	720	22.2	73560	> 1h	720	> 1h	720	16.7
3	360	360	254.7	360	284.8	102900	> 1h	1200	> 1h	360	306.5
4	360	360	98.1	360	91.9	45510	> 1h	—	> 1h	360	94.4
5	720	720	14.3	720	21.0	39600	> 1h	720	> 1h	720	15.5
6	360	360	248.9	360	243.0	103920	> 1h	—	> 1h	360	254.5
7	720	720	20.9	720	21.6	87540	> 1h	720	> 1h	720	29.8
8	180	180	1968.3	180	1959.7	86070	> 1h	—	> 1h	180	2451.5
9	360	360	193.0	360	184.7	81060	> 1h	360	> 1h	360	227.0
10	660	660	24.4	660	27.2	82380	> 1h	660	> 1h	660	19.8
11	480	480	1239.4	480	1122.1	95430	> 1h	—	> 1h	480	911.6
12	900	900	63.2	900	85.3	78420	> 1h	900	> 1h	900	69.8
13	900	900	41.5	900	44.3	124620	> 1h	900	> 1h	900	37.9
14	840	840	25.1	840	33.7	76680	> 1h	840	> 1h	840	30.7
15	480	480	798.8	480	426.9	116310	> 1h	—	> 1h	480	617.9
16	240	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
17	960	960	21.0	960	58.8	77220	> 1h	1440	> 1h	960	41.0
18	840	840	57.9	840	103.0	110400	> 1h	—	> 1h	840	59.4
19	240	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
20	960	960	26.3	960	35.7	90360	> 1h	960	> 1h	960	36.6
21	600	600	1159.7	600	1034.6	152700	> 1h	—	> 1h	600	544.4
22	1080	1080	176.8	1080	1570.6	122280	> 1h	1080	> 1h	1080	407.7
23	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
24	600	600	760.2	600	916.2	140040	> 1h	—	> 1h	600	1043.4
25	600	600	1047.9	600	624.2	138450	> 1h	—	> 1h	600	684.4
26	1020	1020	60.4	1020	84.2	122820	> 1h	—	> 1h	1020	40.5
27	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
28	300	—	> 1h	—	> 1h	—	> 1h	—	> 1h	—	> 1h
29	1140	1140	89.2	1140	358.9	89640	> 1h	1560	> 1h	1140	136.2
30	1020	1020	3316.3	1020	2402.0	173040	> 1h	—	> 1h	1020	754.9

Table 7. Fitness values and runtimes for DataSet2 (WASP)

as well. However, the `oll` approach, based on the same algorithm as clasp's unsatisfiable-core based optimization, is again the fastest WASP configuration on average, thus highlighting the effectiveness of respective optimization methods over complementary implementations.

A general observation made in the solver comparison was that clasp's branch-and-bound based optimization frequently produced more intermediate, non-optimal solutions than WASP with either model- or core-guided approaches. Unfortunately, this also led to the situation that, for the hard instances of DataSet3 and DataSet4 (having no solutions without deviation), WASP was able to provide intermediate solutions for few instances only. We thus omit a corresponding table here, while finding better suited WASP configurations for DataSet3 and DataSet4 would be of interest.

6. Discussion

In this work, we presented a novel approach to tackle the shift design problem by using ASP. Finding good solutions for shift design problems is of great importance in many kinds of organizations. However, such problems are very challenging due to the huge search space and conflicting objectives. Our work contributes to better understanding the strengths of ASP technology in this domain and extends the state of the art for the shift design problem by providing new optimal solutions. Below we summarize the main observations regarding the application of ASP to the shift design problem:

- ASP-based optimization methods show very good results for shift design problems that have solutions without over- and understaffing. Our proposed ASP approach was able to provide optimal solutions for almost all such benchmark instances.
- The results for problems that do not have solutions without over- or understaffing are promising. Although our current approach could not reproduce best known solutions in a number of cases, we were able to find global optima for four hard instances, not previously solved to the optimum.
- Our ASP formulation of the shift design problem is inspired by the order encoding from SAT. The main idea is to express quantitative values in terms of closed intervals to allow for a compact representation of arithmetic comparisons. Such compactness is crucial when facing instances of high scheduling granularity. Given that our benchmark set includes instances with a planning horizon of one week divided into 15-minute slots, in order to avoid memory blow-ups due to grounding, it was important to keep the encoding linear w.r.t. values in the input.
- In our encoding, we also exploited the dual nature of shortage and excess in case they are optimized with common priority and weight. Interestingly, a symmetric treatment significantly improves the quality of intermediate solutions in branch-and-bound based optimization, while it has little effect on unsatisfiable-core based optimization.
- We furthermore showed the flexibility of our approach by providing global optima w.r.t. hierarchically ordered optimization criteria. Distinguishing criteria by priority allows for more targeted optimization, which led to an increased number of instances that could be solved to the optimum.
- The positive effect of domain heuristics on branch-and-bound based optimization indicates that combining our approach with other heuristic methods could be advantageous as well. For example, solutions computed with meta-heuristic or min-cost max-flow techniques may be further improved using ASP.
- Our experimental evaluation showed that ASP-based optimization methods have the potential to provide good solutions for shift design problems. However, finding global optima for large instances without exact solutions is still a challenging task, as there are few hard constraints involved that would help to restrict the search space. Hence, the obtained collection of benchmarks is suitable to assess and further improve state-of-the-art ASP technology.

As future work, we plan to tackle the problem of optimization in shift design by combining ASP with domain-specific heuristics in order to better guide the search. We are confident that ASP combined with

heuristics is a powerful tool for tackling problems in the area of workforce scheduling. This is already underlined by the significantly improved results obtained for the branch-and-bound based approach when activating clasp's integrated heuristics. By using customized heuristics, tailored to the specific problem at hand, the chance for further improvements is thus high.

Acknowledgments. This work was funded by AoF (251170), DFG (550/9), and FWF (P25607-N23, P24814-N23, Y698-N23).

References

[1] Abseher, M.: *Solving Shift Design Problems with Answer Set Programming*, Master Thesis, Technische Universität Wien, 2013.

[2] Abseher, M., Gebser, M., Musliu, N., Schaub, T., Woltran, S.: Shift Design with Answer Set Programming, *Proceedings of the Eighth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'15)* (D. Inclezan, M. Maratea, Eds.), 2015.

[3] Abseher, M., Gebser, M., Musliu, N., Schaub, T., Woltran, S.: Shift Design with Answer Set Programming, *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)* (F. Calimeri, G. Ianni, M. Truszczyński, Eds.), vol. 9345 of *Lecture Notes in Computer Science*, Springer-Verlag, 2015, 32–39.

[4] Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A Native ASP Solver Based on Constraint Learning, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)* (P. Cabalar, T. Son, Eds.), vol. 8148 of *Lecture Notes in Computer Science*, Springer-Verlag, 2013, 54–66.

[5] Alviano, M., Dodaro, C., Marques-Silva, J., Ricca, F.: On the Implementation of Weak Constraints in WASP, *Proceedings of the Seventh Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'14)* (D. Inclezan, M. Maratea, Eds.), 2014.

[6] Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-Based Optimization in clasp, *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)* (A. Dovier, V. Santos Costa, Eds.), vol. 17 of *Leibniz International Proceedings in Informatics*, Dagstuhl Publishing, 2012, 212–221.

[7] Aykin, T.: A Comparative Evaluation of Modeling Approaches to the Labor Shift Scheduling Problem, *European Journal of Operational Research*, **125**(2), 2000, 381–397.

[8] Banbara, M., Soh, T., Tamura, N., Inoue, K., Schaub, T.: Answer Set Programming as a Modeling Language for Course Timetabling, *Theory and Practice of Logic Programming*, **13**(4-5), 2013, 783–798.

[9] Bechtold, S., Jacobs, L.: Implicit Modeling of Flexible Break Assignments in Optimal Shift Scheduling, *Management Science*, **36**(11), 1990, 1339–1351.

[10] Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W.: Scheduling Breaks in Shift Plans for Call Centers, *Proceedings of the Seventh International Conference on the Practice and Theory of Automated Timetabling (PATAT'08)*, 2008.

[11] Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W.: An AI-Based Break-Scheduling System for Supervisory Personnel, *IEEE Intelligent Systems*, **25**(2), 2010, 60–73.

[12] Biere, A., Heule, M., van Maaren, H., Walsh, T., Eds.: *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[13] Brewka, G., Eiter, T., Truszczyński, M.: Answer Set Programming at a Glance, *Communications of the ACM*, **54**(12), 2011, 92–103.

[14] Crawford, J., Baker, A.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems, *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)* (B. Hayes-Roth, R. Korf, Eds.), AAAI Press, 1994, 1092–1097.

[15] Dantzig, G.: A Comment on Eddie's "Traffic delays at toll booths", *Operations Research*, **2**(3), 1954, 339–341.

[16] den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel Scheduling: A Literature Review, *European Journal of Operational Research*, **226**(3), 2013, 367–385.

[17] Di Gaspero, L., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A., Slany, W.: The Minimum Shift Design Problem, *Annals of Operations Research*, **155**(1), 2007, 79–105.

[18] Di Gaspero, L., Gärtner, J., Musliu, N., Schaerf, A., Schafhauser, W., Slany, W.: A Hybrid LS-CP Solver for the Shifts and Breaks Design Problem, *Proceedings of the Seventh International Workshop on Hybrid Metaheuristics (HM'10)* (M. Blesa, C. Blum, G. Raidl, A. Roli, M. Sampels, Eds.), vol. 6373 of *Lecture Notes in Computer Science*, Springer-Verlag, 2010, 46–61.

[19] Di Gaspero, L., Gärtner, J., Musliu, N., Schaerf, A., Schafhauser, W., Slany, W.: Automated Shift Design and Break Scheduling, *Automated Scheduling and Planning – From Theory to Practice* (S. Uyar, E. Özcan, N. Urquhart, Eds.), vol. 505 of *Studies in Computational Intelligence*, Springer-Verlag, 2013, 109–127.

[20] Dodaro, C., Leone, N., Nardi, B., Ricca, F.: Allotment Problem in Travel Industry: A Solution Based on ASP, *Proceedings of the Ninth International Conference on Web Reasoning and Rule Systems (RR'15)* (B. ten Cate, A. Mileo, Eds.), vol. 9209 of *Lecture Notes in Computer Science*, Springer-Verlag, 2015, 77–92.

[21] Faber, W., Leone, N., Perri, S.: The Intelligent Grounder of DLV, *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz* (E. Erdem, J. Lee, Y. Lierler, D. Pearce, Eds.), vol. 7265 of *Lecture Notes in Computer Science*, Springer-Verlag, 2012, 247–264.

[22] Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp Series 3, *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)* (F. Calimeri, G. Ianni, M. Truszczyński, Eds.), vol. 9345 of *Lecture Notes in Computer Science*, Springer-Verlag, 2015, 368–383.

[23] Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-Criteria Optimization in Answer Set Programming, *Technical Communications of the Twenty-seventh International Conference on Logic Programming (ICLP'11)* (J. Gallagher, M. Gelfond, Eds.), vol. 11 of *Leibniz International Proceedings in Informatics*, Dagstuhl Publishing, 2011, 1–10.

[24] Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: *Answer Set Solving in Practice*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.

[25] Gebser, M., Kaminski, R., Schaub, T.: Grounding Recursive Aggregates: Preliminary Report, *Proceedings of the Third Workshop on Grounding, Transforming, and Modularizing Theories with Variables (GTTV'15)* (M. Denecker, T. Janhunen, Eds.), 2015.

[26] Guziolowski, C., Videla, S., Eduati, F., Thiele, S., Cokelaer, T., Siegel, A., Saez-Rodriguez, J.: Exhaustively Characterizing Feasible Logic Models of a Signaling Network Using Answer Set Programming, *Bioinformatics*, **29**(18), 2013, 2320–2326.

[27] Kocabas, D.: *Exact Methods for Shift Design and Break Scheduling*, Master Thesis, Technische Universität Wien, 2015.

[28] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning, *ACM Transactions on Computational Logic*, **7**(3), 2006, 499–562.

[29] Marques-Silva, J., Planes, J.: On Using Unsatisfiability for Solving Maximum Satisfiability, *Computing Research Repository*, **abs/0712.1097**, 2007.

[30] Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and Core-Guided MaxSAT Solving: A Survey and Assessment, *Constraints*, **18**(4), 2013, 478–534.

[31] Musliu, N.: *Intelligent Search Methods for Workforce Scheduling: New Ideas and Practical Applications*, Ph.D. Thesis, Technische Universität Wien, 2001.

[32] Musliu, N., Schaerf, A., Slany, W.: Local Search for Shift Design, *European Journal of Operational Research*, **153**(1), 2004, 51–64.

[33] Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-Prolog Decision Support System for the Space Shuttle, *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)* (I. Ramakrishnan, Ed.), vol. 1990 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001, 169–183.

[34] Quimper, C., Rousseau, L.: A Large Neighbourhood Search Approach to the Multi-Activity Shift Scheduling Problem, *Journal of Heuristics*, **16**(3), 2010, 373–391.

[35] Rekik, M., Cordeau, J., Soumis, F.: Implicit Shift Scheduling with Multiple Breaks and Work Stretch Duration Restrictions, *Journal of Scheduling*, **13**(1), 2010, 49–75.

[36] Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-Building with Answer Set Programming in the Gioia-Tauro Seaport, *Theory and Practice of Logic Programming*, **12**(3), 2012, 361–381.

[37] Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics, *Artificial Intelligence*, **138**(1-2), 2002, 181–234.

[38] Soininen, T., Niemelä, I.: Developing a Declarative Rule Language for Applications in Product Configuration, *Proceedings of the First International Symposium on Practical Aspects of Declarative Languages (PADL'99)* (G. Gupta, Ed.), vol. 1551 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999, 305–319.

[39] Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, **14**(2), 2009, 254–272.

[40] Tellier, P., White, G.: Generating Personnel Schedules in an Industrial Setting Using a Tabu Search Algorithm, *Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)* (E. Burke, H. Rudová, Eds.), vol. 3867 of *Lecture Notes in Computer Science*, Springer-Verlag, 2006, 293–302.

[41] Thompson, G.: Improved Implicit Modeling of the Labor Shift Scheduling Problem, *Management Science*, **41**(4), 1995, 595–607.

[42] Widl, M., Musliu, N.: The Break Scheduling Problem: Complexity Results and Practical Algorithms, *Memetic Computing*, **6**(2), 2014, 97–112.