# Shift Design with Answer Set Programming ⋆

M. Abseher[1], M. Gebser[2,3], N. Musliu[1], T. Schaub[3,4⋆⋆], and S. Woltran[1]

[1] TU Wien, Austria
[2] Aalto University, HIIT, Finland
[3] University of Potsdam, Germany
[4] 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. 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 solving methods based on ASP.

## 1 Introduction

Answer Set Programming (ASP) [11] is a declarative formalism for solving hard computational problems. Thanks to the power of modern ASP technology [18], ASP was successfully used in various application areas, including product configuration [29], decision support for space shuttle flight controllers [25], team building and scheduling [28], and bio-informatics [19]. Recently, ASP also proved successful for optimization problems that had not been amenable to complete methods before, for instance in the domain of timetabling [5].

In this paper, we investigate the application of ASP to another important domain, namely, workforce scheduling [9]. 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

⋆ A short version of this paper will appear at LPNMR'15.
⋆⋆ Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

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 [24], employee requirements for a period of time, constraints about the possible start and length 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 [24] and a hybrid method combining network flow with local search [14]. 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 several 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 [12] can be used in ASP for modeling complex interval constraints. On the other hand, our empirical evaluation contrasts traditional model-guided[5] optimization techniques with orthogonal core-guided techniques [21], revealing another case in which the latter have an edge over the former. Although our experiments show that the shift design problem provides challenging benchmarks for state-of-the-art ASP technology, we were 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 [24]. As input, we are given the following:

- consecutive *time slots* sharing the same length. Each time slot is associated with a number of employees that should be present during the slot.
- *shift types* with associated parameters *min-start* and *max-start*, representing the earliest and latest start, and *min-length* and *max-length*, representing the minimum and maximum length of a shift. An example of such shift types is given in Table 1.

The aim is to generate a collection of $k$ shifts $s_1, \ldots, s_k$. Each shift $s_i$ is completely determined by its *start* and *length*, which must belong to some shift type. Additionally, each shift $s_i$ is associated with parameters indicating the number of employees assigned to $s_i$ during each day 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.

[5] That is, branch-and-bound based strategies; the term 'model-guided' was coined in [22, 2].

| *shift type* | *min-start* | *max-start* | *min-length* | *max-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

In analogy to [14], we investigate the optimization of the following criteria: sum of *shortages* of workers in each time slot during the planning period, sum of *excesses* of workers in each time slot during the planning period, and the *number of shifts*.[6] Traditionally, the objective function 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 [24], 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 different companies. The complexity of the shift design problem was analyzed in [14], 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 [8, 7, 16, 33]. An detailed overview of previous work on shift design and break scheduling is given in [15].

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 [13], several formulations were proposed in [4, 6, 27, 32]. Other approaches include large neighborhood search [26], tabu search [31], etc. The original shift design problem differs in several aspects from the shift scheduling problem. In our problem, 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.

Finally, there is also the area of workforce management where the focus is on the allocation of employees of different qualifications to tasks requiring different skills. Similar to shift design and scheduling problems, constraints concerning the workload have to be taken into account. In [28], 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 encoded problem of [28] is a classical allocation problem with optimization towards work balance, while the problem we tackle here aims at an optimal alignment of shifts.

## 3 Shift Design in ASP

An instance like the one shown in Figure 1 is specified by facts as in Figure 2. Facts of the form $time(S,T)$ associate each slot $S$ with a day time $T$. Our instance includes

[6] In [24], additionally, the average number of duties per week is considered.

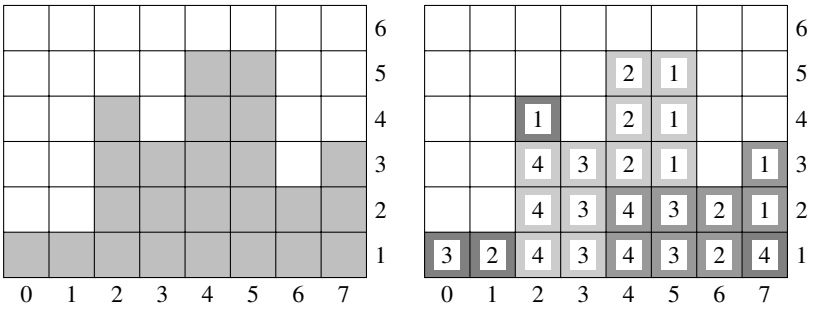


**Fig. 1.** Work demands over a day (left) and the unique optimal schedule (right) with shifts starting at slot 2, 4, or 7, respectively, indicated by different boxes, while other kinds of shifts are unused

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

**Fig. 2.** ASP facts specifying an instance of the shift design problem

one day, divided into eight slots denoted by the times $0, \ldots, 7$. Instances of $\mathit{next}(S', S)$ provide predecessor or successor slots, respectively, where $S$ is usually $S'+1$, except for the last slot whose successor is 0. (When another day is added, the slots $8, \ldots, 15$ would also be mapped to day times $0, \ldots, 7$, $\mathit{next}(7,0)$ would be replaced by $\mathit{next}(7,8)$, and $\mathit{next}(15,0)$ would connect the new last slot to 0 instead.) For each slot $S$, a fact $\mathit{work}(S,N)$ gives the number $N$ of desired employees, and $\mathit{exceed}(E)$ as well as $\mathit{shorten}(F)$ may limit the amount of employees at duty to at most $E+N$ or at least $N-F$, respectively. For instance, we obtain the upper bound 4 and the lower bound 2 for employees engaged in slot 7. Facts of the form $\mathit{range}(S,L,1), \ldots, \mathit{range}(S,L,M)$ provide potential amounts of shifts of length $L$ that can start from slot $S$, where $M$ is the maximum number of desired employees over all slots within the horizon of the shift. For shifts starting from slot 7, those of length 2 or 3 stretch to slot 0 or 1, respectively, and the corresponding maximum number of desired employees is 3 in slot 7 itself; unlike that, shifts of length 4 also include slot 2 in which 4 employees shall be at duty.

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 a priority $P$ and a penalty weight $W$ incurred in case of violations. The priorities in Figure 2 tell us that 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 in terms of day time and length should be as small as feasible. Given that the criteria are already distinguished by priority, the

$$\{run(S,L,I)\} \leftarrow range(S,L,I) \quad (1)$$
$$run(S,L,I) \leftarrow run(S',L{+}1,I), next(S',S), 0 < L \quad (2)$$
$$run(S,L,I) \leftarrow run(S,L{+}1,I), 0 < L \quad (3)$$
$$run(S,L,I{+}J) \leftarrow run(S,L{+}1,I), shift(S,L,J) \quad (4)$$
$$\leftarrow run(S,L,I{+}1), 0 < I, {\sim}run(S,L,I) \quad (5)$$
$$\leftarrow work(S,N), exceed(E), run(S,1,N{+}E{+}1) \quad (6)$$
$$\leftarrow work(S,N), shorten(F), F < N, {\sim}run(S,1,N{-}F) \quad (7)$$
$$length(S,L,I,1) \leftarrow range(S,L,I), run(S,L,I), {\sim}run(S,L{+}1,I) \quad (8)$$
$$length(S,L,I,J) \leftarrow length(S,L,I{+}1,J{-}1), 0 < I, {\sim}run(S,L{+}1,I) \quad (9)$$
$$shift(S,L,J) \leftarrow length(S,L,I,J) \quad (10)$$
$$shift(S,L,J) \leftarrow shift(S',L{+}1,J), next(S',S), 0 < L \quad (11)$$
$$start(S,L,J) \leftarrow range(S,L,J), next(S',S), shift(S,L,J), {\sim}shift(S',L{+}1,J) \quad (12)$$
$$W@P,S,I,shortage \leftsquigarrow opt(shortage,P,W), work(S,N), I \in [1,N), {\sim}run(S,1,I) \quad (13)$$
$$W@P,S,I,excess \leftsquigarrow opt(excess,P,W), work(S,N), run(S,1,I), N < I \quad (14)$$
$$W@P,T,L,select \leftsquigarrow opt(select,P,W), start(S,L,J), time(S,T) \quad (15)$$

**Fig. 3.** ASP encoding of the shift design problem

penalty weight of a violation of either kind is 1, thus counting particular violations to assess schedules.

Our ASP encoding of the shift design problem is shown in Figure 3. For a slot $S$, the intuitive reading of the predicate $run(S,L,I)$ is that at least $I$ shifts including $S$ and $L{-}1$ or more successor slots are scheduled. This is further refined by $length(S,L,I,J)$, telling that $1,\ldots,J$ of the scheduled shifts of exact length $L$ may start from $S$, where $I{-}1$ shifts that include at least $L{-}1$ successor slots are scheduled in addition. The predicate $shift(S,L,J)$ expresses that at least $J$ of the scheduled shifts stretch to $S$ and exactly $L{-}1$ successor slots, and $start(S,L,J)$ indicates that the $J$-th instance of such a shift indeed starts from $S$. A schedule is thus characterized by the number of (true) atoms of the form $start(S,L,J)$, yielding the amount of shifts of length $L$ 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 $start(7,4,1)$. Respective residual lengths, 4 at the start of each instance of a shift and then decremented at successor slots, are given in the center of boxes indicating the scheduled shifts, and associated positions are reflected by the height at which boxes are placed. At slots 2, 4, and 7, the positions happen to coincide with the number of shifts starting from them, viz. three, two, or one instance of length 4, respectively. In particular, atoms $shift(2,4,1)$, $shift(2,4,2)$, and $shift(2,4,3)$, derived in view of $length(2,4,1,3)$, $length(2,4,2,2)$, and $length(2,4,3,1)$, express that three shifts of exact residual length 4 are scheduled at slot 2. After decreasing the residual length to 2 at slot 4, the information about scheduled shifts is used to relocate the three instances to the positions 3, 4, and 5, as two shifts with a longer residual length than 2 start from slot 4. In fact, the five shifts scheduled at slot 4 are represented by $run(4,4,1)$, $run(4,4,2)$, $run(4,2,3)$, $run(4,2,4)$, and $run(4,2,5)$ along with additional atoms ob-

tained by propagating residual lengths, 4 or 2, respectively, down to 1. Also note that a comparison between shifts scheduled at slots 3 and 4 yields that only the instances of residual length 4 start from 4. Indeed, 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 slot 2, 4, or 7, respectively.

In more detail, the potential start of an instance $I$ of a shift of length $L$ from slot $S$ is reflected by the choice rule (1) in Figure 3. Rule (2) propagates the start of a shift to its $L-1$ successor slots, where the residual length is decreased down to 1 in the last slot of the shift. For shifts with longer residual length $L$, rule (3) closes the interval between 1 and $L$, thus overturning any choice rules for potential starts of shifts of shorter length. Moreover, this allows for pushing the $J$-th instance of a shift stretching to slot $S$ to the position $I+J$ when $I$ instances of shifts longer than the residual length $L$ are scheduled, as expressed by rule (4). The integrity constraint (5) asserts that the positions associated with scheduled shifts must be ordered by residual length. This condition eliminates guesses on instances $I$ of starting shifts, and it also provides a shortcut making interconnections between positions of scheduled shifts explicit, which turned out as effective to improve search performance. The additional integrity constraints (6) and (7) are applicable whenever the deviation from numbers of desired employees is bounded above or below, respectively. Note that it is sufficient to inspect atoms of the form $\mathit{run}(S,1,I)$ for appropriate positions $I$, given that residual lengths are propagated via rule (3).

In order to derive the amount of scheduled shifts of exact residual length $L$, rule (8) marks positions $I$, where instances may start, with 1 when the length $L$ matches. Instances associated with smaller positions then count on by means of rule (9) unless their positions are occupied by shifts with longer residual length. By projecting the positions out, rule (10) yields that $1,\ldots,J$ shifts of length $L$ may start from slot $S$. In addition, longer shifts whose residual length decreases to $L$ in $S$ are propagated via rule (11). Finally, rule (12) compares instances that may start to propagated shifts and indicates the ones that indeed start from $S$. As a consequence, a stable model represents a schedule in terms of sequences of the form $\mathit{start}(S,L,m),\ldots,\mathit{start}(S,L,n)$, expressing that $n+1-m$ instances of a shift of length $L$ start from slot $S$. It remains to assess the quality of a schedule, which is accomplished by means of the weak constraints (13), (14), and (15) for the three optimization criteria at hand. The penalty for deviating from a number of desired employees is characterized in terms of the priority $P$ and weight $W$ given in facts, a position $I$ pointing to under- or overstaffing in a slot $S$, and the 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 slot $S$ of a starting shift of length $L$ to its day time $T$, so that the penalty $W@P$ is incurred at most once for a shift with particular parameters, no matter how many instances are actually utilized.

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 instances of shifts. The basic idea is similar to the so-called *order encoding* [12], which has been successfully applied to solve constraint satisfaction problems by means of SAT [30]. In our ASP encoding, rule (4), (8), and (9) 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 scheduled. Likewise, the integrity constraints (6) and (7) as well as the weak constraints (13) and (14) focus on value 1, standing for any residual length, to determine the amount of employees at duty. That is, the order encoding approach enables a compact formulation of existence tests and general conditions, which then propagate to all target values greater or smaller than a certain threshold.

## 4 Experiments

In the following, we present the experimental evaluation of our approach. 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 7.8 (wheezy). Each test run, using Clingo 4.4.0 [17], was bound to a single core and 8GB RAM with a time limit of 60 minutes. Preliminary tests showed best results with Clingo's configuration `handy`. Additionally, we also consider the configuration `tweety` as it is the default for ASP problems. To support the search for optimal solutions, we use the two different optimization strategies of Clingo: (i) Branch-and-bound based with hierarchical steps (`--opt-strategy=bb,1`) and (ii) Unsatisfiable-core based with disjoint-core preprocessing and implications instead of equivalences (`--opt-strategy=usc,3`). For strategy (i), we also considered domain heuristics (`--dom-mod=4,8`) in order to accelerate the process of convergence towards the optimum.

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 Clingo allows for non-uniform descents during optimization. For instance in multi-criteria optimization, this enables the consideration of criteria in the order of significance, rather than producing spurious (intermediate) solutions.

*Core-guided* approaches originated in the area of MaxSAT [10]. They rely on successively identifying and relaxing unsatisfiable cores until a solution that is guaranteed to be optimal is obtained (see [21]). The (main) implementation in Clingo relies on the core-guided optimization algorithm *oll* [3]. It can (optionally) be combined with disjoint-core preprocessing [20], which as a side effect provides a quick approximation of the optimum (otherwise no intermediate solutions are obtained).

*Problem instances* The benchmarks are grouped into three different sets of instances of the shift design problem. Below we briefly explain their basic structure and most important characteristics. All benchmark sets are publicly available under the address `http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html`.[7] The data sets were first described in [23, 24] and also used in [14] 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 workers to a selected number of shifts (also called the *seed solution*), and then the resulting values were used as requirements in respective instances.

[7] Instances as facts and our ASP encoding are available at:
`http://www.dbai.tuwien.ac.at/proj/Rota/DataSetASP.zip`

*DataSet2* The second data set contains 30 instances, which are quite similar to those of the first data set, but here the seed solution was constructed in such a way that instances 1–10 should need at least 12 shifts to be solved exactly. The instances 11–20 feature 16 shifts, and the remaining ten instances were constructed with a seed solution using 20 shifts. Di Gaspero et al. [14] note that their heuristic could also find better solutions for some of the problem instances. In our experimental evaluation, we thus use their results for reference. The second data set was originally constructed with the intention to study the impact of the number of shifts in the best known solution on computation time.

*DataSet3* Di Gaspero et al. [14] 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. These invalid shifts cannot be selected in an optimal 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 a seed of 12 shifts (valid and invalid ones), and also the remaining instances are generated using the same scheme concerning the number of shifts as in the second data set.

*DataSet4* The fourth set contains three advanced 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 length of a time slot 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.

*Evaluation* Tables 2, 3, and 4 illustrate the fitness values and runtimes for all data sets. In the second column of the tables, the currently best known fitness values are provided, and the columns to the right of it show the results we obtained using ASP.

All values in the tables represent the median for five test runs at the time of their termination for each instance and configuration. The reference values in columns for the best known fitness taken from [14] represent the mean over 10, for DataSet1 and DataSet2, or 100, in case of DataSet3 and DataSet4, trials with incomplete methods. An entry "> 1h" in a runtime column denotes that the corresponding instance could not be solved within the time limit of one hour. A dash in a column for the fitness means that Clingo did not produce any solution within one hour. Since all our experiments are conducted without a limit on the maximum shortage and excess and because we also do not restrict the maximum number of shifts, we can directly compare our results to previous work on these instances, and we will provide formerly unknown global optima for four instances. In fact, the fitness in [14] is based on a balanced sum penalizing deviations and the utilization of shifts equitably, so that all weak constraints are of the same priority and penalty 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 (given in columns headed by `--opt-strategy=usc,3`). Non-

| Instance | Best Fitness [14] | `--opt-strategy=bb,1` | | | | `--opt-strategy=bb,1 --dom-mod=4,8` | | | | `--opt-strategy=usc,3` | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | `tweety` | | `handy` | | `tweety` | | `handy` | | `tweety` | | `handy` | |
| | | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time |
| 1 | 480 | 53640 | > 1h | 60960 | > 1h | 2820 | > 1h | 3780 | > 1h | 480 | 7.9 | 480 | 14.7 |
| 2 | 300 | 22410 | > 1h | 24990 | > 1h | 3000 | > 1h | 6000 | > 1h | 300 | 80.4 | 300 | 80.1 |
| 3 | 600 | 44280 | > 1h | 45840 | > 1h | 5160 | > 1h | 5460 | > 1h | 600 | 13.2 | 600 | 23.2 |
| 4 | 450 | 71370 | > 1h | 62160 | > 1h | 11730 | > 1h | 11310 | > 1h | 450 | 625.3 | 450 | 203.3 |
| 5 | 480 | 26340 | > 1h | 24300 | > 1h | 480 | 606.4 | 1500 | > 1h | 480 | 4.5 | 480 | 7.1 |
| 6 | 420 | 23160 | > 1h | 19620 | > 1h | 420 | 108.4 | 420 | 513.8 | 420 | 2.3 | 420 | 3.3 |
| 7 | 270 | 68880 | > 1h | 77670 | > 1h | 2640 | > 1h | 5040 | > 1h | 270 | 105.1 | 270 | 102.8 |
| 8 | 150 | 97590 | > 1h | 94560 | > 1h | 18015 | > 1h | 18735 | > 1h | — | > 1h | 150 | 3482.5 |
| 9 | 150 | 27075 | > 1h | 25995 | > 1h | 10155 | > 1h | 6525 | > 1h | 150 | 1731.2 | 150 | 1573.1 |
| 10 | 330 | 73680 | > 1h | 71310 | > 1h | 9390 | > 1h | 7800 | > 1h | 330 | 127.8 | 330 | 124.1 |
| 11 | 30 | 25575 | > 1h | 27375 | > 1h | 30 | 770.7 | 30 | 3351.8 | 30 | 208.0 | 30 | 211.4 |
| 12 | 90 | 47460 | > 1h | 49980 | > 1h | 4740 | > 1h | 2955 | > 1h | 90 | 884.4 | 90 | 922.0 |
| 13 | 105 | 53295 | > 1h | 51570 | > 1h | 8625 | > 1h | 7170 | > 1h | 105 | 1463.7 | 105 | 1528.5 |
| 14 | 195 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 15 | 180 | 6840 | > 1h | 180 | 543.6 | 180 | 3.6 | 180 | 11.3 | 180 | 0.5 | 180 | 0.6 |
| 16 | 225 | 70365 | > 1h | 71445 | > 1h | 17880 | > 1h | 19275 | > 1h | 225 | 3178.4 | 225 | 3281.4 |
| 17 | 540 | 88440 | > 1h | 90720 | > 1h | 11040 | > 1h | 9510 | > 1h | 540 | 295.7 | 540 | 306.7 |
| 18 | 720 | 42840 | > 1h | 42960 | > 1h | 7140 | > 1h | 7620 | > 1h | 720 | 12.7 | 720 | 22.5 |
| 19 | 180 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 20 | 540 | 42300 | > 1h | 30300 | > 1h | 540 | 3146.2 | 2580 | > 1h | 540 | 5.6 | 540 | 9.2 |
| 21 | 120 | 29910 | > 1h | 30540 | > 1h | 5640 | > 1h | 5430 | > 1h | 120 | 982.1 | 120 | 983.3 |
| 22 | 75 | 36420 | > 1h | 35550 | > 1h | 1500 | > 1h | 1455 | > 1h | 75 | 499.8 | 75 | 492.8 |
| 23 | 150 | 49530 | > 1h | 42645 | > 1h | 14100 | > 1h | 13245 | > 1h | 150 | 2210.8 | 150 | 2018.9 |
| 24 | 480 | 26940 | > 1h | 20940 | > 1h | 480 | 525.9 | 480 | 2942.7 | 480 | 3.4 | 480 | 5.5 |
| 25 | 480 | 94200 | > 1h | 86790 | > 1h | 11520 | > 1h | 11190 | > 1h | 480 | 348.9 | 480 | 307.9 |
| 26 | 600 | 41640 | > 1h | 23580 | > 1h | 1440 | > 1h | 3180 | > 1h | 600 | 7.1 | 600 | 12.1 |
| 27 | 480 | 51360 | > 1h | 52560 | > 1h | 6840 | > 1h | 5220 | > 1h | 480 | 11.8 | 480 | 20.6 |
| 28 | 270 | 18390 | > 1h | 22470 | > 1h | 2970 | > 1h | 2670 | > 1h | 270 | 25.4 | 270 | 37.5 |
| 29 | 360 | 62010 | > 1h | 56310 | > 1h | 10170 | > 1h | 9000 | > 1h | 360 | 128.7 | 360 | 141.7 |
| 30 | 75 | 16875 | > 1h | 17055 | > 1h | 1215 | > 1h | 2385 | > 1h | 75 | 308.4 | 75 | 326.5 |

**Table 2.** Fitness values and runtimes for DataSet1

surprisingly, we observed that using branch-and-bound based optimization leads to a vast amount of intermediate, non-optimal solutions, so that fitness values are improved rather slowly. In contrast, the unsatisfiable-core based approach consumes more time to come up with intermediate solutions, but it produces much fewer of them before converging to an optimum. An important point to mention is the fact that using domain heuristics in addition to the branch-and-bound approach (results given in columns including `--dom-mod=4,8`) is significantly improving on branch-and-bound without domain heuristics.

Apart from the fact that `--opt-strategy=usc,3` delivers global optima for almost all 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. Consider for example Instances 3 and 4 in Table 2, where the `tweety` configuration turns out to be better than `handy` for Instance 3 with both optimization strategies. On the other hand, for Instance 4, it leads to a three times longer run with `--opt-strategy=usc,3` and significantly deteriorates the fitness value with `--opt-strategy=bb,1`. Due to the very similar nature of the second data set, we make the same observations there, as shown in Table 3.

| Instance | Best Fitness [14] | `--opt-strategy=bb,1` | | | | `--opt-strategy=bb,1 --dom-mod=4,8` | | | | `--opt-strategy=usc,3` | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | `tweety` | | `handy` | | `tweety` | | `handy` | | `tweety` | | `handy` | |
| | | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time | Fitness | Time |
| 1 | 720 | 39540 | > 1h | 19680 | > 1h | 3120 | > 1h | 4020 | > 1h | 720 | 5.1 | 720 | 8.36 |
| 2 | 720 | 63240 | > 1h | 33600 | > 1h | 4680 | > 1h | 6180 | > 1h | 720 | 10.8 | 720 | 15.8 |
| 3 | 360 | 82560 | > 1h | 79620 | > 1h | 13320 | > 1h | 11730 | > 1h | 360 | 231.0 | 360 | 246.0 |
| 4 | 360 | 42780 | > 1h | 43140 | > 1h | 5880 | > 1h | 6540 | > 1h | 360 | 81.4 | 360 | 82.9 |
| 5 | 720 | 29820 | > 1h | 23100 | > 1h | 5220 | > 1h | 5340 | > 1h | 720 | 6.8 | 720 | 10.8 |
| 6 | 360 | 78750 | > 1h | 70500 | > 1h | 14160 | > 1h | 12630 | > 1h | 360 | 207.1 | 360 | 213.3 |
| 7 | 720 | 62880 | > 1h | 43200 | > 1h | 7500 | > 1h | 8460 | > 1h | 720 | 16.6 | 720 | 17.6 |
| 8 | 180 | 65505 | > 1h | 67200 | > 1h | 10845 | > 1h | 9480 | > 1h | 180 | 1729.4 | 180 | 1742.0 |
| 9 | 360 | 68130 | > 1h | 62880 | > 1h | 11340 | > 1h | 9570 | > 1h | 360 | 163.9 | 360 | 168.8 |
| 10 | 660 | 67320 | > 1h | 35340 | > 1h | 5580 | > 1h | 7560 | > 1h | 660 | 12.7 | 660 | 21.3 |
| 11 | 480 | 55830 | > 1h | 56280 | > 1h | 9750 | > 1h | 8250 | > 1h | 480 | 1012.5 | 480 | 1003.4 |
| 12 | 900 | 62940 | > 1h | 38400 | > 1h | 5460 | > 1h | 7320 | > 1h | 900 | 27.0 | 900 | 41.0 |
| 13 | 900 | 56820 | > 1h | 31980 | > 1h | 10620 | > 1h | 9900 | > 1h | 900 | 41.9 | 900 | 32.9 |
| 14 | 840 | 69060 | > 1h | 35460 | > 1h | 8040 | > 1h | 9420 | > 1h | 840 | 18.8 | 840 | 21.1 |
| 15 | 480 | 87450 | > 1h | 94380 | > 1h | 13620 | > 1h | 11070 | > 1h | 480 | 697.7 | 480 | 683.4 |
| 16 | 240 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 17 | 960 | 53820 | > 1h | 43860 | > 1h | 7320 | > 1h | 7860 | > 1h | 960 | 19.8 | 960 | 19.2 |
| 18 | 840 | 62040 | > 1h | 58200 | > 1h | 8760 | > 1h | 10560 | > 1h | 840 | 34.3 | 840 | 52.4 |
| 19 | 240 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 20 | 960 | 66600 | > 1h | 36600 | > 1h | 8280 | > 1h | 10140 | > 1h | 960 | 23.8 | 960 | 21.3 |
| 21 | 600 | 112530 | > 1h | 101460 | > 1h | 14940 | > 1h | 15030 | > 1h | 600 | 633.6 | 600 | 964.0 |
| 22 | 1080 | 105240 | > 1h | 60360 | > 1h | 11220 | > 1h | 11640 | > 1h | 1080 | 353.7 | 1080 | 128.1 |
| 23 | 300 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 24 | 600 | 119910 | > 1h | 113700 | > 1h | 12780 | > 1h | 11850 | > 1h | 600 | 1147.2 | 600 | 706.5 |
| 25 | 600 | 107970 | > 1h | 84750 | > 1h | 16410 | > 1h | 15060 | > 1h | 600 | 2297.1 | 600 | 974.6 |
| 26 | 1020 | 100440 | > 1h | 46860 | > 1h | 8340 | > 1h | 7800 | > 1h | 1020 | 42.3 | 1020 | 53.2 |
| 27 | 300 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 28 | 300 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 29 | 1140 | 76800 | > 1h | 31140 | > 1h | 10020 | > 1h | 11160 | > 1h | 1140 | 147.7 | 1140 | 77.8 |
| 30 | 1020 | 91620 | > 1h | 68640 | > 1h | 13260 | > 1h | 9960 | > 1h | 1020 | 1887.8 | 1020 | 2347.3 |

**Table 3.** Fitness values and runtimes for DataSet2

While these outcomes already show that at least `--opt-strategy=usc,3` is a viable choice for 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 want 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 [14]. All in all, although the shift design problem turns out to be challenging for state-of-the-art ASP solvers, our evaluation highlights the usefulness of ASP in the domain of workforce scheduling.

## 5 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 different organizations. However, such problems are very challenging due to the huge

| Instance | Best Fitness [14] | --opt-strategy=bb,1 | | | | --opt-strategy=bb,1 --dom-mod=4,8 | | | | --opt-strategy=usc,3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | tweety | | handy | | tweety | | handy | | tweety | | handy | |
| | | 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 | 60600 | > 1h | 63720 | > 1h | 21330 | > 1h | 18780 | > 1h | 12930 | > 1h | 12840 | > 1h |
| 3 | 9582.14 | 45840 | > 1h | 46110 | > 1h | 20640 | > 1h | 18780 | > 1h | **9540** | **3338.2** | **9540** | **1948.6** |
| 4 | 6634.40 | 87240 | > 1h | 91860 | > 1h | 18480 | > 1h | 17160 | > 1h | 8700 | > 1h | **6540** | **1680.9** |
| 5 | 9996.00 | 82020 | > 1h | 34920 | > 1h | 16800 | > 1h | 16140 | > 1h | 13500 | > 1h | 13380 | > 1h |
| 6 | 2076.75 | 57945 | > 1h | 59430 | > 1h | 9135 | > 1h | 9105 | > 1h | 5445 | > 1h | 6300 | > 1h |
| 7 | 6075.00 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 8 | 8860.50 | 64500 | > 1h | 64050 | > 1h | 19320 | > 1h | 19440 | > 1h | 12660 | > 1h | 12870 | > 1h |
| 9 | 6036.90 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 10 | 2968.95 | 48240 | > 1h | 47100 | > 1h | 10500 | > 1h | 10140 | > 1h | 5940 | > 1h | 6810 | > 1h |
| 11 | 5490.90 | 97530 | > 1h | 79530 | > 1h | 21570 | > 1h | 20760 | > 1h | 9840 | > 1h | 10200 | > 1h |
| 12 | 4171.20 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 13 | 4662.00 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 14 | 9616.55 | 53520 | > 1h | 40020 | > 1h | 14820 | > 1h | 15480 | > 1h | 12900 | > 1h | 12720 | > 1h |
| 15 | 11445.00 | 72090 | > 1h | 69510 | > 1h | 24060 | > 1h | 24780 | > 1h | 13530 | > 1h | 13650 | > 1h |
| 16 | 10734.00 | 55200 | > 1h | 31740 | > 1h | 20460 | > 1h | 18120 | > 1h | 13800 | > 1h | 13560 | > 1h |
| 17 | 4729.05 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 18 | 6692.40 | 61890 | > 1h | 66300 | > 1h | 16650 | > 1h | 14280 | > 1h | 10440 | > 1h | 10950 | > 1h |
| 19 | 5157.45 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 20 | 9153.90 | 84420 | > 1h | 89250 | > 1h | 28110 | > 1h | 24420 | > 1h | 15540 | > 1h | 16920 | > 1h |
| 21 | 6053.55 | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h | — | > 1h |
| 22 | 12870.30 | 94890 | > 1h | 97950 | > 1h | 29640 | > 1h | 26880 | > 1h | 15750 | > 1h | 15600 | > 1h |
| 23 | 8384.24 | 88440 | > 1h | 85680 | > 1h | 19440 | > 1h | 18840 | > 1h | 13320 | > 1h | 12660 | > 1h |
| 24 | 10417.80 | 66900 | > 1h | 63180 | > 1h | 23460 | > 1h | 22980 | > 1h | 13800 | > 1h | 15720 | > 1h |
| 25 | 13204.80 | 100560 | > 1h | 85740 | > 1h | 23520 | > 1h | 19740 | > 1h | **13020** | **41.2** | **13020** | **44.1** |
| 26 | 13117.80 | 95760 | > 1h | 94620 | > 1h | 37680 | > 1h | 33420 | > 1h | 22020 | > 1h | 20610 | > 1h |
| 27 | 10081.20 | 96360 | > 1h | 61440 | > 1h | 19020 | > 1h | 19200 | > 1h | **10020** | **101.4** | **10020** | **1138.5** |
| 28 | 10603.80 | 68040 | > 1h | 46200 | > 1h | 20160 | > 1h | 20040 | > 1h | 14580 | > 1h | 13800 | > 1h |
| 29 | 6690.00 | 89670 | > 1h | 92130 | > 1h | 21870 | > 1h | 22290 | > 1h | 11010 | > 1h | 10290 | > 1h |
| 30 | 13723.80 | 39480 | > 1h | 59520 | > 1h | 20820 | > 1h | 23400 | > 1h | 17460 | > 1h | 17520 | > 1h |
| DataSet4 | | | | | | | | | | | | | |
| 1 | N/A | 67920 | > 1h | 44700 | > 1h | 51600 | > 1h | 50040 | > 1h | 57600 | > 1h | 60060 | > 1h |
| 2 | N/A | 77010 | > 1h | 86880 | > 1h | 16860 | > 1h | 19320 | > 1h | 13260 | > 1h | 13170 | > 1h |
| 3 | N/A | 164760 | > 1h | 135780 | > 1h | 36360 | > 1h | 35280 | > 1h | 25980 | > 1h | 28560 | > 1h |

**Table 4.** Fitness values and runtimes for DataSet3 and DataSet4

search space and conflicting constraints. 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 shows very good results for shift design problems that have solutions without over- and understaffing. Our proposed ASP approach could provide optimal solutions for almost all such benchmark instances.
- The first results for problems that do not have solutions without over- or understaffing are promising. Although our current approach could not reproduce best known solutions for several problems, we were able to provide global optima for four hard instances, not previously solved to the optimum.

– Our experimental evaluation indicates that our approach could also be used in combination with other search techniques. For example, solutions computed by meta-heuristic methods or min-cost max-flow techniques could be further improved by ASP.
– In general, the computational results show that ASP has the potential to provide good solutions in this domain. Therefore, our results open up the area of workforce scheduling, which is indeed challenging for state-of-the-art ASP solvers. This is most probably caused by the nature of the shift design problem, as there are few hard constraints involved that could help to restrict the search space.

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, but also exploiting Clingo's integrated features is a promising target for further investigation. We are confident that ASP combined with heuristics is a powerful tool for tackling problems in the area of workforce scheduling. This fact is already underlined by the significantly improved results obtained for the branch-and-bound based approach when activating Clingo'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's thesis, Technische Universität Wien (2013)
2. Alviano, M., Dodaro, C., Marques-Silva, J., Ricca, F.: On the implementation of weak constraints in wasp. In: Proceedings of ASPOCP'14 (2014)
3. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Technical Communications of ICLP'12, pp. 212–221. LIPIcs (2012)
4. Aykin, T.: A comparative evaluation of modeling approaches to the labor shift scheduling problem. European Journal of Operational Research 125(2), 381–397 (2000)
5. 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), 783–798 (2013)
6. Bechtold, S., Jacobs, L.: Implicit modeling of flexible break assignments in optimal shift scheduling. Management Science 36(11), 1339–1351 (1990)
7. Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W.: Scheduling breaks in shift plans for call centers. In: Proceedings of PATAT'08 (2008)
8. Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W.: An AI-based break-scheduling system for supervisory personnel. IEEE Intelligent Systems 25(2), 60–73 (2010)
9. den Bergh, J., Beliën, J., Bruecker, P., Demeulemeester, E., Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research 226(3), 367–385 (2013)
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press (2009)
11. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)

12. Crawford, J., Baker, A.: Experimental results on the application of satisfiability algorithms to scheduling problems. In: Proceedings of AAAI'94, pp. 1092–1097. AAAI Press (1994)
13. Dantzig, G.: A comment on Eddie's "Traffic delays at toll booths". Journal of the Operations Research Society of America 2(3), 339–341 (1954)
14. Di Gaspero, L., Gärtner, J., Kortsarz, G., Musliu, N., Schaerf, A., Slany, W.: The minimum shift design problem. Annals of Operations Research 155(1), 79–105 (2007)
15. Di Gaspero, L., Gärtner, J., Musliu, N., Schaerf, A., Schafhauser, W., Slany, W.: Automated shift design and break scheduling. In: Automated Scheduling and Planning, pp. 109–127. Springer (2013)
16. 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. In: Proceedings of HM'10, pp. 46–61. Springer (2010)
17. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 105–124 (2011)
18. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool Publishers (2012)
19. 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), 2320–2326 (2014)
20. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. Computing Research Repository (CoRR), 0712.1097 (2007)
21. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints 18(4), 478–534 (2013)
22. Morgado, A., Heras, F., Marques-Silva, J.: Model-guided approaches for MaxSAT solving. In: Proceedings of ICTAI'13, pp. 931–938. IEEE Computer Society (2013)
23. Musliu, N.: Intelligent search methods for workforce scheduling: New ideas and practical applications. Ph.D. thesis, Technische Universität Wien (2001)
24. Musliu, N., Schaerf, A., Slany, W.: Local search for shift design. European Journal of Operational Research 153(1), 51–64 (2004)
25. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-Prolog decision support system for the space shuttle. In: Proceedings of PADL'01, pp. 169–183. Springer (2001)
26. Quimper, C., Rousseau, L.: A large neighbourhood search approach to the multi-activity shift scheduling problem. Journal of Heuristics 16(3), 373–392 (2010)
27. Rekik, M., Cordeau, J., Soumis, F.: Implicit shift scheduling with multiple breaks and work stretch duration restrictions. Journal of Scheduling 13(1), 49–75 (2010)
28. 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), 361–381 (2012)
29. Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: Proceedings of PADL'99, pp. 305–319. Springer (1998)
30. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints 14(2), 254–272 (2009)
31. Tellier, P., White, G.: Generating personnel schedules in an industrial setting using a tabu search algorithm. In: Proceedings of PATAT'06, pp. 293–302 (2006)
32. Thompson, G.: Improved implicit modeling of the labor shift scheduling problem. Management Science 41(4), 595–607 (1995)
33. Widl, M., Musliu, N.: The break scheduling problem: Complexity results and practical algorithms. Memetic Computing 6(2), 97–112 (2014)