TEMPLE - A Domain Specific Language for Modeling and Solving Staff Scheduling Problems

Johannes Gärtner
Ximes GmbH
Hollandstraße 12/12
A-1020 Vienna, Austria
gaertner@ximes.com

Nysret Musliu
Vienna University of Technology
Favoritenstraße 9-11
A-1040 Vienna, Austria
musliu@dbai.tuwien.ac.at

Werner Schafhauser
Ximes GmbH
Hollandstraße 12/12
A-1020 Vienna, Austria
schafhauser@ximes.com

Wolfgang Slany
Graz University of Technology
Inffeldgasse 16B/II
A-8010 Graz, Austria
wsi@ist.tugraz.at

***Abstract*—We present TEMPLE, a domain specific language for modeling and solving staff scheduling problems. TEMPLE provides a set of intuitive abstractions and notations allowing to formulate the constraints of a particular problem in a very compact and natural way. After modeling a staff scheduling problem in TEMPLE, three generic local search algorithms can immediately be applied to the corresponding optimization problem. We show how real-life staff scheduling problems can be both effectively modeled as well as efficiently solved using our approach. Finally, we report on a practical application of TEMPLE in a commercial staff scheduling software.**

I. Introduction

Staff scheduling problems represent very difficult tasks of high practical relevance. In the field of staff scheduling we usually have to design shift plans satisfying several, often conflicting criteria, e.g., legal demands, labor rules and staffing requirements. Obtaining optimal or close to optimal solutions for staff scheduling problems improves the working conditions for employees and helps companies to deploy their staff efficiently and cost savingly.

For instance Figure 5 presents a typical staff scheduling problem. As input we are given staffing requirements (dashed line), specifying for each point in time a minimum number of employees that should be working. The solution of that problem consists of a shift plan (gray shaded horizontal bars) and a break schedule (white blocks) satisfying both legal demands and staffing requirements. Further examples of staff scheduling problems can be found in [1], [2], [4], [12], [13].

Unfortunately, many staff scheduling problems are NP-hard and, as a consequence, they cannot be solved to optimality in polynomial time. Therefore, staff scheduling problems are solved using mixed-integer-programming or sophisticated AI-techniques, such as constraint programming, heuristics, meta-heuristic methods, or branch and bound algorithms. No matter which of these approaches we follow, the design of algorithms for staff scheduling problems is an art in itself and we end up with a solution that is strongly customized to a specific staff scheduling problem. Customized solutions are usually very difficult to adapt, extend and reuse for other staff scheduling problems. A few minor changes within a problem's specification can result in many major modifications or a completely new implementation of a customized algorithm.

To overcome these drawbacks we present the domain specific programming language TEMPLE. TEMPLE has been developed to model and solve staff scheduling problems with reduced effort. For that purpose the following design goals are realized by TEMPLE:

Modularity: A TEMPLE problem consists of small, concise building blocks reflecting common features of staff scheduling problems. New building blocks are derived from already existing ones. By this principle users are forced to formulate a complex problem in small, concise and traceable steps. Consequently, the resulting problem models are well-structured, easy to understand, modify and maintain.

Adaptability and Extensibility: Problems modeled in TEMPLE can be adapted easily. A few small changes in the problem formulation result only in a few small changes in the corresponding TEMPLE program.

Simplicity: TEMPLE requires only basic programming skills from its users. Anybody familiar with a third generation programming language should be able to understand and use TEMPLE.

Automatic Optimization: Once a problem is modeled in TEMPLE it can be optimized in an instant without requiring additional coding from the user.

Efficiency: TEMPLE's intrinsic computational overhead is kept as little as possible. Thereby we ensure that problems cannot only be modeled effectively but also solved efficiently with TEMPLE.

To achieve automatic optimization we present a TEMPLE compiler which transforms the TEMPLE model of a staff scheduling problem into three executable local search algorithms: a simulated annealing algorithm, a hill-climbing based approach, and an iterated local search algorithm. These algorithms can be applied instantaneously and do not require any further user input or modifications. To ensure efficiency the obtained local search algorithms recompute and update only those parts of a solution which are actually affected by changes.

To demonstrate TEMPLE's modularity and simplicity we model and solve a complex real-life break scheduling problem [3] in TEMPLE. The resulting TEMPLE program consists of only 500 lines of code and is written in a very concise, understandable and modular manner. We evaluate the iterated

978-1-61284-196-0//11/$26.00 ©2011 IEEE

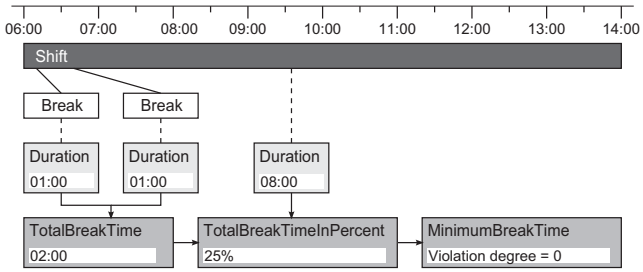

Fig. 1. In TEMPLE properties and constraints are derived step by step from already existing properties.

local search algorithm obtained with TEMPLE on ten publicly available benchmark instances for our problem. Our computational results reveal that TEMPLE is able to compute solutions of acceptable quality.

Finally, we successfully adapt and extend the previous TEMPLE program in order to solve a multilayer staff scheduling task. The obtained TEMPLE program represents the core of a commercial staff scheduling tool which is already applied successfully by decision makers in their day-to-day business.

II. The Modeling Language TEMPLE

A. Intervals and Links

A central building block of staff scheduling problems are intervals. Shifts, breaks, meetings or absence times can be considered as intervals. Moreover, different kinds of intervals are linked with each other, e.g., a shift is linked with the breaks scheduled in it and vice versa. For that reason TEMPLE enables the declaration of different kinds of intervals and links in the following manner:

```
Interval Shift;     //Interval declarations.
Interval Break;
Shift <-> Break;    //Declaration of a bi-directional link.
```

An interval is characterized by four basic properties, `Start`, `Duration`, `End`, and a boolean basic property `Active`, indicating whether an interval is part of a problem's solution or not.

B. Derived Elements

A characteristic of staff scheduling problems is that their features and constraints can be derived step by step, one after the other. For instance, in many real-life applications it is common to require that a minimum percentage of break time, e.g., 20%, must be scheduled in each shift. Figure 1 shows how that constraint can be derived for a single shift in several steps. First of all we compute the break time scheduled in the shift by summing up the duration of its breaks. Then we determine the break time percentage and finally we impose the constraint, checking whether the shift's break time percentage exceeds 20%. In TEMPLE we realized the concept of derived elements to model features and constraints of staff scheduling problems exactly in that manner. On the basis of already existing properties we derive additional properties and finally we can impose constraints.

1) Derived Properties: In TEMPLE a derived property reflecting a shift's total break time (Figure 1) might be formulated as follows:

```
Property Shift::TotalBreakTime(Shift.Break[] scheduledBreak)
{
  TotalBreakTime =
    sum(i in scheduledBreak.getRange())
                (scheduledBreak[i].Duration);
}
```

This code snippet specifies that each shift has an additional property called `TotalBreakTime`. That property is derived from all breaks linked to a single shift `Shift.Break[]`, which can be accessed through the alias `scheduledBreak`. The value of property `TotalBreakTime` is computed by summing up the durations of each break scheduled within the shift.

2) Derived Constraints: We distinguish between two kinds of constraints: hard constraints and soft constraints. Hard constraints specify the criteria which must be satisfied completely by any feasible solution. Except for the keyword `HardConstraint` the violation degree of a hard constraint is derived in the same manner as the value of a derived property.

```
HardConstraint Shift::MinimumBreakTime(Shift thisShift)
{
  if(thisShift.TotalBreakTimeInPercent < 20)
    MinimumBreakTime = VIOLATED;
}
```

Soft constraints on the other hand model the objectives to be reached by a good solution. The importance of a soft constraint within an entire staff scheduling problem is expressed in terms of integer weights.

3) Derived Curves: Curves are central building blocks of staff scheduling problems, which can be used to model many features of staff scheduling tasks such as staffing requirements or available staff. Figure 2 presents how the time periods during that an employee is actually working and not having a break can be represented as a curve over time.

In TEMPLE we can derive curves from intervals or previously defined curves by using a predefined set of curve operations. These operations increment or decrement a curve over a certain period, they write or read a value at a specific position, or they add and subtract other, already existing, curves. For instance, the curve representing an employee's actual working time can be modeled in the following way:

```
Curve Shift::WorkingTime(Shift thisShift,
                         Shift.Break[] scheduledBreak)
{
  //Increment curve from shift start until shift end.
  WorkingTimePattern.Pulse( thisShift.Start,
                            thisShift.End,
                            thisShift.Active);

  forall(i in scheduledBreak.getRange())
  {
    //Decrement curve along each break.
    WorkingTimePattern.Pulse( scheduledBreak[i].Start,
                              scheduledBreak[i].End,
                              scheduledBreak[i].Active, -1);
  }
}
```

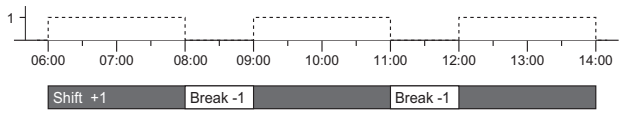

Fig. 2. A curve modeling the periods while an employee is actually working and not having a break.

C. Initial Solution

After we have modeled the structure of a particular staff scheduling problem by the help of intervals, links, derived properties, curves and constraints, we have to specify an initial solution for a particular staff scheduling problem. In TEMPLE the initial solution is formulated in three different ways. First of all, in each TEMPLE program we specify an input XML-file. That XML-file contains a list of intervals and stores the initial basic properties of each interval. Secondly, we can force the instantiation of further intervals. Thirdly, we may compute and assign initial values to the basic interval properties `Start`, `Duration` and `Active`. The initial values are derived from already existing properties or curves of linked intervals. Furthermore, we can also restrict the domains of basic properties and we may introduce additional links between intervals.

```
Initialize Shift::BreakSchedule(Shift thisShift,
                                Shift.Break[] breakToSchedule)
{
  forall(i in breakToSchedule.getRange())
  {
    //1. Assign initial values to basic break properties.
    breakToSchedule[i].Start    = thisShift.Start;
    breakToSchedule[i].Duration = 30 minutes;
    breakToSchedule[i].Active   = true;

    //2. Restrict domains of a break's start and duration.
    forall(j in thisShift.Start .. thisShift.End)
      breakToSchedule[i].Start.Domain.Add(j);

    breakToScheduled[i].Duration.Domain.Add(30 minutes);

    //3. Link the shift with each break scheduled within it.
    breakToSchedule[i].AddLink(thisShift, "Shift");
  }
}
```

D. Moves

Local search algorithms try to improve the quality of a current solution by applying small changes. In terms of local search techniques these changes are called moves. To define a move in TEMPLE we compute and assign new values for the basic interval properties and assign them to these properties. For instance, the following code sample specifies a move placing a break at a new position in its associated shift.

```
Move Shift::PutBreakAtNewPosition(Shift thisShift,
                                  Shift.Break[] scheduledBreak)
{
  range S = thisShift.Start .. thisShift.End;

  select(i in scheduledBreak.getRange())
    select(newPosition in S)
      scheduledBreak[i].Start = newPosition;
}
```

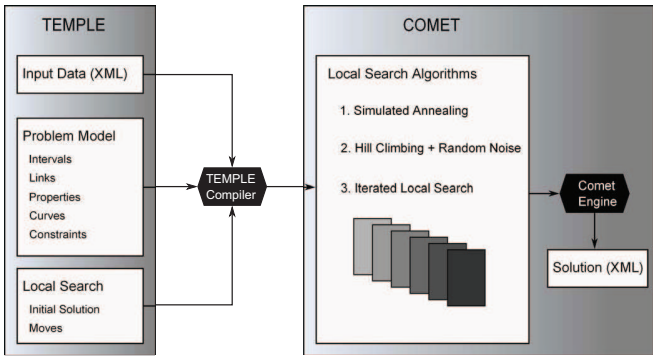

Fig. 3. Solving staff scheduling problems in TEMPLE.

E. Further Language Details

For the sake of completeness we describe which additional information we must specify to obtain a compilable TEMPLE program: an input XML-file containing input intervals and initial basic property values, a solution XML-file in which the obtained solution of a problem shall be saved, the local search algorithm which shall be applied to a particular problem, a limit on the algorithm running time and the granularity of the planning period.

```
input                   = "./input_data.xml";
solution                = "./solution.xml";
algorithm               = iterated local search;
algorithm running time  = 1 minute;
time slot               = 10 minutes;
```

III. The TEMPLE Compiler

We implemented a TEMPLE compiler to transform TEMPLE programs into executable local search algorithms that solve the underlying staff scheduling problems. As input our compiler is passed the problem model formulated in the TEMPLE modeling language and an XML-file containing input information of a particular problem instance. On the basis of that input the TEMPLE compiler generates three local search algorithms for the considered staff scheduling tasks: a simulated annealing algorithm, a hill climbing strategy and an iterated local search algorithm. The three local search algorithms are written in the constraint-based optimization language COMET [10]. To obtain a solution for the considered staff scheduling problem the generated algorithms are executed by the COMET optimization engine. Finally, the best solution found during the execution of a local search algorithm is returned as an XML-file. Figure 3 illustrates the entire approach we followed to solve staff scheduling problems in TEMPLE.

A. Optimization Goal

In TEMPLE we use hard and soft constraints to define the optimization goals of a considered staff scheduling problem. If S denotes the set of all soft constraints and H the set of all hard constraints defined in a particular TEMPLE program, the local search algorithms generated by our TEMPLE compiler try to solve the following optimization problem:

$$min \quad \sum_{s \in S} s.Weight \times s.ViolationDegree$$

$$s.t. \quad \forall h \in H : h.ViolationDegree = 0$$

B. Initialization

Every local search algorithm must create an initial solution at its beginning. Therefore, the three local search algorithms generated by our TEMPLE compiler have to compute initial values for each derived property, curve and constraint. Since derived elements depend on each other, they cannot be initialized in any arbitrary order. If we reconsider the derived elements presented in Figure 1 we see that first we have to initialize property `TotalBreakTime`, then property `TotalBreakTimeInPercent` and finally constraint `MinimumBreakTime`.

To ensure that the initial solution is obtained correctly, the TEMPLE compiler analyzes a given TEMPLE model and determines the dependencies between properties, curves and constraints. On the basis of that analysis the compiler builds a dependency graph and determines a feasible initialization ordering of all derived elements. The compiler encodes that ordering in the resulting local search algorithms and in that manner it is guaranteed that the initial solution is computed correctly.

C. Efficient Local Search

Although there are significant differences among the local search algorithms generated by the TEMPLE compiler they basically apply the same three main steps in each iteration:

1) They compute a set of moves to obtain a local neighbourhood of the current solution.
2) They evaluate the effect of each move on the current solution. When evaluating a move they check whether the move is feasible, i.e., it does not cause any hard constraint violations, and they determine the difference in the problem's objective function resulting from the move.
3) They select a feasible move and execute it to obtain a new solution.

Most computational effort is spent on the evaluation and execution of moves. To ensure that these two steps are carried out efficiently, we apply the following strategies in the local search algorithms created by the TEMPLE compiler:

Lazy Evaluation: If we observe that a move violates a hard constraint we will not evaluate the move's effect on other hard and soft constraints.

Caching: We use a move cache to store the result of each evaluation of a move on a property, curve or constraint. With that move cache we can avoid that a move is evaluated several times for the same derived element.

Efficient Move Evaluation and Execution: When evaluating a move's effect on a solution we only evaluate those properties, curves and constraints that are affected by the move. Similarly, when performing a move we update only those solution elements which are actually changed by a move.

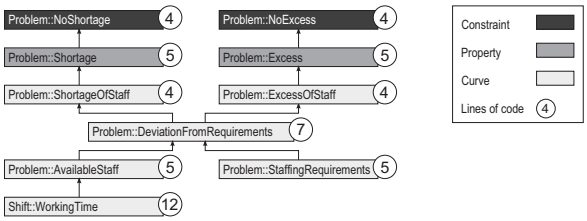

Fig. 4. TEMPLE model of selected constraints of a real-life staff scheduling problem [3].

Efficient Data Structures: To evaluate a move's effect on curves efficiently we developed and implemented a speed-up strategy. This strategy ensures that only those curve positions are evaluated which are affected by a move. By applying that speed-up strategy we could reduce the computational costs associated with curves significantly.

IV. Practical Application of TEMPLE

A. Solving a Real-Life Staff Scheduling Problem

To demonstrate that staff scheduling problems can be modeled elegantly in TEMPLE and to evaluate the performance of the local search algorithms generated with TEMPLE we consider a real-life break scheduling problem originating in the area of supervision personnel [3]. As input we are given the staffing requirements over an entire week, an already designed shift plan and five constraints specifying how much break time must be scheduled and how breaks must be placed in a legal break schedule. Two further constraints require that shortage and excess of staff should be reduced to a minimum degree.

1) The TEMPLE Model: We succeeded in modeling the break scheduling problem in TEMPLE. In our model the five criteria concerning the legality of the break pattern (constraints C_1 - C_5 in [3]) are formulated as hard constraints, whereas shortage and excess of staff (constraints C_6 and C_7 in [3]) are modeled as soft constraints. The initial solution for our problem is obtained with the heuristic proposed in [3] involving simple temporal problems (STPs) [5]. The applied heuristic constructs an initial break pattern satisfying all hard constraints. Finally, we specified two moves, the first one places a break at a new position in its shift (see Section II-D), the second move swaps two breaks of different duration.

In total we needed one man-week to develop a suitable TEMPLE model for the considered break scheduling problem. The resulting TEMPLE program consists only of 500 lines of code and it is formulated in a very modular style. Figure 4 visualizes how we modeled the two soft constraints requiring that shortage and excess of staff shall be reduced to a minimum degree within a good solution. For each employee we introduce a curve representing the actual working time. By summing up all these single curves we obtain a curve modeling the available staff. In the next step we subtract the staffing requirements from the available staff and so we obtain the deviations from staffing requirements. Then we extract the negative deviations to obtain a curve representing the shortage of staff, determine the total amount shortage associated with

	No. of shifts violating constraint					Time Slots with			
				Min.		Shortage		Excess	
Instance	Break positions	Lunch breaks	Work periods	break times	Break durations	No.	%	No.	%
2fc04a	0	0	0	0	0	99	4.9	664	32.9
2fc04a03	0	0	0	0	0	91	4.5	673	33.4
2fc04a04	0	0	0	0	0	78	3.9	754	37.4
3fc04a	0	0	0	0	0	97	4.8	445	22.1
3fc04a03	0	0	0	0	0	82	4.1	466	23.1
3fc04a04	0	0	0	0	0	68	3.4	560	27.8
3si2ji2	0	0	0	0	0	8	0.4	1082	53.7
4fc04a	0	0	0	0	0	87	4.3	450	22.3
4fc04a03	0	0	0	0	0	80	4.0	477	23.7
4fc04a04	0	0	0	0	0	64	3.2	562	27.9

TABLE I
FEATURES OF THE BEST SOLUTIONS OBTAINED WITH TEMPLE-ILS FOR TEN BENCHMARK INSTANCES.

a solution, and finally we impose a constraint requiring that shortage of employees should be avoided. The constraint penalizing excess of employees is derived in a similar manner.

The circled numbers in Figure 4 indicate how many lines of code are needed to formulate a particular property, curve or constraint. Only a few lines of code, twelve at maximum, are necessary to derive each single element. On average only eleven lines of code are needed to model a single TEMPLE element in the entire program.

2) Quality of Obtained Solutions: With our TEMPLE compiler we created an iterated local search algorithm from our TEMPLE model for the break scheduling problem for supervisory personnel. This algorithm will be denoted TEMPLE-ILS in the remainder of this section. To evaluate TEMPLE-ILS we applied it to ten benchmark instances for the considered break scheduling task which are available at www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks. We executed algorithm TEMPLE-ILS on the same machine under the same conditions as the algorithm presented in Beer et al. [3], namely on a Genuine Intel T2400 laptop running at 1.8 GHz with 2 Gbytes of RAM, with a one hour time limit on each single run.

Table I presents the features of the best solutions obtained by TEMPLE-ILS in ten runs for each benchmark instance. All hard constraints concerning the labor rules and legal requirements are satisfied completely. Considering soft constraints, according to Beer et al. [3] in a good solution for the considered instances shortage of staff is below a five percent threshold. This criterion could be met for each of the ten considered benchmark instances by the solutions obtained with TEMPLE-ILS. Figure 5 presents a part of the best solutions obtained for benchmark instance 2fc04a by TEMPLE-ILS. In the considered period shortage of staff is avoided completely and, as in all solutions obtained by TEMPLE-ILS, all constraints concerning the legality of the break pattern are satisfied completely. Thus, we conclude that with our TEMPLE model of the break scheduling problem for supervisory personnel we are able to compute solutions of acceptable quality in reasonable time.

B. Commercial Application of TEMPLE

TEMPLE has already been applied successfully in a commercial staff scheduling tool. We built this tool in a research project together with an industrial partner. The goal of the project was to develop a working prototype used by decision makers on-site to solve a multilayer staff scheduling problem.

In this staff scheduling problem we are given task requirements for an entire day, an already existing shift plan and the qualifications of each employee. To obtain a solution we must again compute a break schedule which is completely consistent with a set of legal requirements. In addition, we must also assign the required tasks to available employees in accordance with their qualifications. Furthermore, a good task assignment must satisfy several criteria. For instance, each task must be performed by the same employee at least for thirty minutes and employees should not be forced to change the task they carry out.

Since the considered problem is very complex as a whole we decided to decompose it into three separate phases each of which is modeled and solved by a separate TEMPLE program:

I. **Break Schedule Initialization.** We compute a legal break schedule which is consistent with all constraints imposed on a break pattern, training units and intra-day absences.
II. **Break Schedule Optimization.** We optimize the break schedule according to staffing requirements, whereby we ensure that the break schedule remains always legal during and after this optimization phase.
III. **Task Assignment and Optimization.** For each time slot we assign the required tasks to available employees heuristically. Afterwards we further try to reduce the violations of constraints imposed on the task assignment to a minimum degree.

The TEMPLE program for the break scheduling problem in Section IV-A could be easily extended and adapted to solve the first and the second phase of the decomposed problem. Therefore, we only needed to concentrate on developing an accurate TEMPLE program for the third phase. By modeling tasks as intervals linked with employees we also succeeded in modeling phase three in a very natural way, and eventually we obtained the desired prototype solving the entire staff scheduling problem. With that prototype we delivered a proof of concept that automated break scheduling and task assignment was possible within a reasonable amount of time, i.e., approximately five minutes on a state of the art computer. The prototype has been extended into a commercial application, which is already used successfully by decision makers in their day-to-day business.

V. RELATED MODELING LANGUAGES

In the last three decades several languages have been developed for modeling combinatorial optimization problems including OPL [9], ASPEN [8], COMET [10], ESRA [6], ESSENCE [7] and Zinc [11].

ESRA, ESSENCE and Zinc were developed to specify or model general combinatorial problems. One important difference is that, unlike TEMPLE, ESRA and ESSENCE do not allow user-defined properties or constraints. Moreover, all

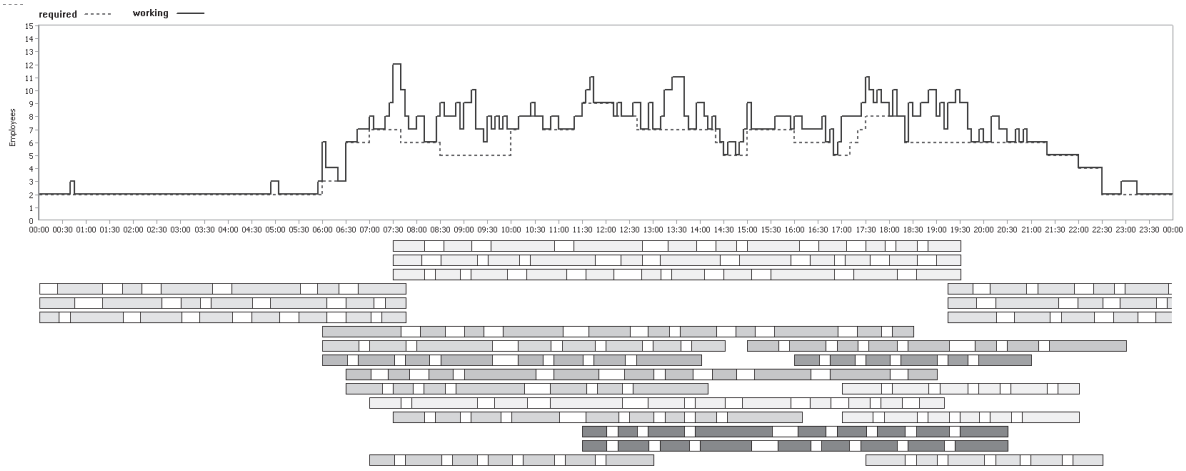

Fig. 5. Part of the best solution found for the real-life instance 2fc04a.

three languages do not offer any staff scheduling specific data structures like intervals or curves.

ASPEN is an application framework for scheduling tasks arising in the field of spacecraft operations. In ASPEN, a problem model consists of activities to be performed and constraints imposed on activities and other solution elements. Although some aspects of ASPEN are akin to the TEMPLE language, e.g., activities are similar to intervals, ASPEN is strongly focused on the characteristics of space craft operations and cannot be applied directly to staff scheduling problems.

As for COMET, TEMPLE inherited some of its syntax and data structures, e.g., sets, ranges and selectors, and COMET is the target language of our TEMPLE compiler. However, TEMPLE is focused of staff scheduling and offers abstractions and notations, such as intervals, links and curves, reflecting common features of staff scheduling tasks. Since we wanted TEMPLE to be as simple as possible, no knowledge about object orientation or any details on local search techniques is required from a user.

OPL [9] was developed to combine the advantages of mathematical programming and constraint programming and has been realized within the software IBM ILOG CPLEX Optimization Studio. Like TEMPLE, that software provides further language elements to facilitate the development of scheduling models such as intervals and cumulative functions which are similar to curves. However, in OPL the user must use a fixed set of temporal or specialized constraints which cannot be extended further whereas in TEMPLE arbitrary constraints can be defined. Moreover, in IBM ILOG OPL we may only adjust some parameters of the underlying exact search method but a user cannot provide additional information regarding the search process as it is possible in TEMPLE by specifying moves.

VI. Conclusion

In this article we presented TEMPLE, a constraint-based language for modeling and solving staff scheduling problems. We highlighted the key concepts and elements of the TEMPLE modeling language, namely intervals, derived elements, initialization and move definitions, which enable the natural and compact formulation of staff scheduling tasks. Moreover, we implemented a TEMPLE compiler to map a TEMPLE model to three executable local search algorithms. Thereby we solved several technical challenges to guarantee that local search algorithms are initialized correctly and executed efficiently. Furthermore, we evaluated TEMPLE by modeling and solving a complex real-life staff scheduling task. For the considered problem TEMPLE was able to return competitive results and solutions of acceptable quality. Finally we reported that TEMPLE is already used successfully within a commercial staff scheduling tool.

VII. Acknowledgments

The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (Bundesministerium für Wirtschaft und Arbeit), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna's Center for Innovation and Technology (ZIT).

References

[1] T. Aykin. Optimal shift scheduling with multiple break windows. *Managament Science*, 42(4):591–602, 1996.

[2] S. E. Bechtold and L. W. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339 –1351, 1990.

[3] A. Beer, J. Gärtner, N. Musliu, W. Schafhauser, and W. Slany. An AI-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25(2):60–73, 2010.

[4] J. Brusco and L. Jacobs. A simulated annealing approach to the cyclic staff-scheduling problem. *Naval Research Logistics*, 40:69–84, 1993.

[5] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[6] P. Flener, J. Pearson, and M. Ågren. Introducing ESRA, a relational language for modelling combinatorial problems. In *CP*, page 971, 2003.
[7] A. M. Frisch, W. Harvey, C. Jefferson, B. M. Hernández, and I. Miguel. ESSENCE: A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, 2008.
[8] A. Fukunaga, G. Rabideau, S. Chien, and A. Govindjee. ASPEN: An application framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS97), Tokyo, Japan*, pages 181–187, 1997.
[9] P. V. Hentenryck. *The OPL Optimization Language*. The MIT-Press, Cambridge, Mass., 1999.
[10] P. V. Hentenryck and L. Michel. *Contraint-Based Local Search*. The MIT Press, Cambridge, Mass., 2005.
[11] K. Marriott, N. Nethercote, R. Rafeh, P. J. Stuckey, M. G. de la Banda, and M. Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.
[12] N. Musliu, A. Schaerf, and W. Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51–64, 2004.
[13] M. Rekik, J.-F. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13(1):49–75, 2010.