A Hybrid LS-CP Solver for the Shifts and Breaks Design Problem

Luca Di Gaspero[1], Johannes Gärtner[2], Nysret Musliu[3], Andrea Schaerf[1], Werner Schafhauser[3], and Wolfgang Slany[4]

[1] DIEGM, Università degli Studi di Udine, Italy
{l.digaspero,schaerf}@uniud.it
[2] Ximes Inc., Austria
gaertner@ximes.com
[3] DBAI, Technische Universität Wien, Austria
{musliu,schafha}@dbai.tuwien.ac.at
[4] IST, Technische Universität Graz, Austria
wolfgang.slany@tugraz.at

Abstract. The problem of designing workforce shifts and break patterns is a relevant employee scheduling problem that arises in many contexts, especially in service industries. The issue is to find a minimum number of shifts, the number of workers assigned to them, and a suitable number of breaks so that the deviation from predetermined workforce requirements is minimized.
We tackle this problem by means of a hybrid strategy in the spirit of Large Neighborhood Search, which exploits a set of Local Search operators that resort to a Constraint Programming model for assigning breaks. We test this strategy on a set of random and real life instances employed in the literature.

1 Introduction

The typical process for scheduling of employees in an organization consists of several stages. Usually the first phase is to find the temporal requirements, which determine the needed number of employees for each day and time interval during the planning period. For example in the staffing requirements given in Fig. 1a there should be 6 employees at work every Monday between 08:00-11:00. After the temporal requirements are defined, the shifts can be designed. In this stage the number of employees for each shift and day has to be determined. Moreover, for each employee assigned to a shift, a set of breaks that fulfill different constraints about their location and lengths should be scheduled (Fig. 1b). Fig. 2 presents possible shifts and scheduled breaks for the instance in Fig. 1.

Shift and break assignment problems may differ in constraints and objectives and are referred in the literature with different names as shift design [9,13], shift scheduling [3,4,17,19], and break scheduling [5,14,20]. Such problems arise in airports, call centers, and service industry in general and have been extensively investigated in Operations Research and recently have been also tackled with

AI techniques. Obtaining good solutions for shift design and break scheduling is of vital relevance due to legal issues considering the working time of employees, well-being of employees, and the importance of reducing costs while satisfying staffing requirements. In some working environments it is imperative to obtain good solutions, e.g., assigning inadequate breaks to air traffic controllers can lead to loss of concentration which can cause serious problems.

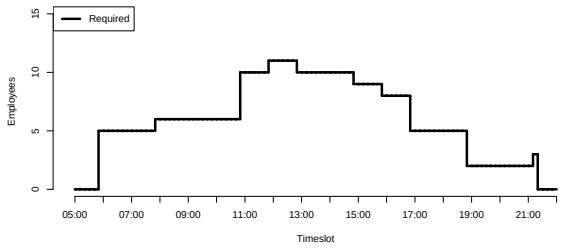
Both shift design and break scheduling are characterized with a huge search space and many conflicting constraints that have to be satisfied. Due to the complexity of these problems, two approaches have been followed in the literature. If the number of breaks to be scheduled is not large (2-3 breaks) the design of shifts and break assignment is performed simultaneously. In cases when the number of breaks to be assigned per shift is large (up to ten breaks) the shift design and break scheduling problems are solved separately, i.e. break assignment is performed only after shifts are generated. Decomposing the general problem (shift design and break scheduling) into two sub-problems makes it easier to solve, but it is hard to predict which shift plan will produce good solutions after break assignments. Therefore, this approach usually requires the knowledge of domain experts when the shift plan is generated. The solution of the entire problem when the number of breaks to be assigned is large is a challenging task due to much larger search space of possible solutions.

We investigate a solution procedure of the whole problem (shift design and break scheduling) with a large number of breaks. Our contribution lies on automating the whole process of shift design and break assignment, and therefore minimizing the need for a domain expert in this phase of staff scheduling. To the best of our knowledge the whole shift design and break scheduling problem we consider in this paper has not been yet investigated in the literature. To this aim, we propose a new hybrid Local Search-Constraint Programming (LS-CP) method to solve the problem. The method have been applied and experimentally evaluated on a set of real life and randomly generated instances.

2 Problem Definition

It is useful to define an *interval* as a structure composed of two attributes, `start` and `length`, which describe the temporal interval $[\texttt{start}, \texttt{start} + \texttt{length})$. In the following, interval variables will be denoted by greek letters. We also use basic interval arithmetic operations such as the sum of a value to the interval, i.e., $a \oplus [b, c) = [a + b, a + c)$.

For the shift and break design problem we are given a set D of days, which are subdivided into a set of n equally long consecutive timeslots $\mathcal{T} = \{\tau_1 = [a_1, a_2), \tau_2 = [a_2, a_3), \ldots, \tau_n = [a_n, a_{n+1})\}$, at a given time granularity. Each timeslot τ_t belongs (entirely) to day $\lfloor t/g \rfloor$, where g is the time granularity measured as the number of timeslots per day. Moreover, for each timeslot $\tau_t \in \mathcal{T}$, we are given a *staffing requirement* r_t, which indicates the number of employees that should be working during that timeslot (see Fig. 1a for an example).



(a) Workforce requirements

Name	Earliest start	Latest start	Minimum length	Maximum length
Morning Shift	05:00	08:00	07:00	09:00
Day Shift	09:00	12:00	07:00	09:00
Evening Shift	13:00	15:00	07:00	09:00

(b) Shift types

Fig. 1: An example instance.

There are given also a set of p shift types $\{v_1, \ldots, v_p\}$. Each shift type v_j constrains the earliest and the latest start ($v_j.\texttt{min\_start}$ and $v_j.\texttt{max\_start}$) and the minimum and maximum length ($v_j.\texttt{min\_length}$ and $v_j.\texttt{max\_length}$) for the shifts belonging to that type (see Fig. 1b for an example).

The problem consists in designing the shifts and deciding the break patterns, i.e., determining the values of the following decision variables for each shift s_i:

(V_1) The shift type $y_i \in \{v_1, \ldots, v_p\}$.

(V_2) The shift interval σ_i. On the basis of its shift type, the allowed values for σ_i are as follows: $\sigma_i.\texttt{start} \in [y_i.\texttt{min\_start}, y_i.\texttt{max\_start}], \sigma_i.\texttt{length} \in [y_i.\texttt{min\_length}, y_i.\texttt{max\_length}]$. It is important to observe that the determined σ_i interval will be the same throughout the time horizon, i.e., each shift is constrained to have precisely the same start time (e.g., 05:00) and length (e.g., 08:00) on all the days.

(V_3) The number of employees w_{id} assigned to shift s_i on day d.

(V_4) For each employee e working on day d, the set of break intervals $\mathcal{B}_{ide} = \{\beta^1_{ide}, \beta^2_{ide}, \ldots\}$; for each $b = 1, \ldots, |\mathcal{B}_{ide}|$, the interval variables have the following domains $\beta^b_{ide}.\texttt{start} \in (d \cdot g) \oplus \sigma_i$ and $\beta^b_{ide}.\texttt{length} \in [1, \sigma_i.\texttt{length}]$.

It is worth noticing that, according to the formulation, also the number of breaks $|\mathcal{B}_{ide}|$ for each shift/day/employee has to be determined (although its value is constrained between lower and upper bounds that can be computed by reasoning on break patterns). This is because the legal working/break patterns do depend also on the number of breaks, and together they do affect the objective function of the problem.

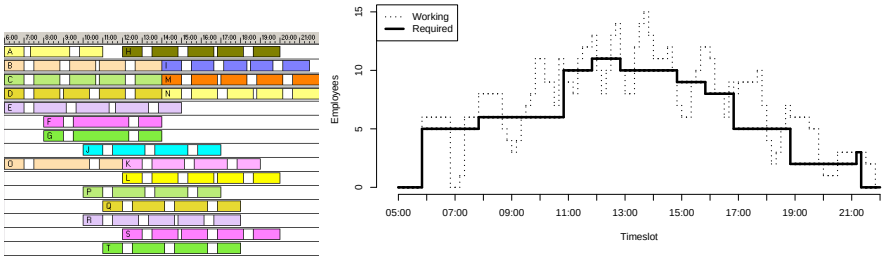


Fig. 2: A possible solution of the problem in Fig. 1.

An employee is considered to be working during the timeslots comprised in the shift but not in any of his/her breaks in that shift. More formally, an employee e works on timeslot $\tau_t \in \mathcal{T}$ on day $d = \lfloor t/g \rfloor$, if the following condition holds: $t \in (d \cdot g) \oplus \sigma_i \setminus \bigcup_{b=1}^{|\mathcal{B}_{ide}|} \beta_{ide}^b$. This condition can also be employed to define the so called *working periods* ω_{ide}^b, $b = 1, \ldots, |\mathcal{B}_{ide}| + 1$, which are the complementary intervals of the shift w.r.t. the breaks.

The following constraints on the break patterns must hold:

HC_1: A fixed total amount of break time, depending on the length of the shift, must be granted to every worker, i.e., $\sum_{b=1}^{|\mathcal{B}_{ide}|} \beta_{ide}^b.\texttt{length} = \texttt{total\_break\_time}_i$, for $d \in D, e = 1, \ldots, w_{id}$.

HC_2: The break length must be included within two bounds, i.e., $\texttt{break\_minimum\_length} \leq \beta_{ide}^b.\texttt{length} \leq \texttt{break\_maximum\_length}$.

HC_3: If the working period preceding a break is too long, the break minimum length is altered, i.e., $\omega_{ide}^{b-1} \geq \texttt{long\_working\_period} \Rightarrow \beta_{ide}^b.\texttt{length} \geq \texttt{long\_break\_minimum\_length}$; obviously, $\texttt{break\_minimum\_length} \leq \texttt{long\_break\_minimum\_length} \leq \texttt{break\_maximum\_length}$.

HC_4: The length of the working periods must be included within two bounds, i.e., $\texttt{working\_period\_minimum\_length} \leq \omega_{ide}^b.\texttt{length} \leq \texttt{working\_period\_maximum\_length}$.

HC_5: The first and the last break have to start at a given distance from the shift extremes, i.e., $\beta_{ide}^1.\texttt{start} \geq \texttt{earliest\_break\_start}$ and $\beta_{ide}^{|\mathcal{B}_{ide}|}.\texttt{start} \leq \texttt{latest\_break\_start}$.

HC_6: If a shift exceeds the length `min_length_for_lunch`, it must include a lunch break λ_{ide}, whose length is fixed (i.e., $\lambda_{ide}.\texttt{length} = \texttt{lunch\_break\_length}$) and its start must lay at a given distance from the shift extremes, that is, $\lambda_{ide}.\texttt{start} \geq \texttt{earliest\_lunch\_break\_start}$ and $\lambda_{ide}.\texttt{start} \leq \texttt{latest\_lunch\_break\_start}$.

The objective for the shift design and break scheduling problem is to generate a feasible solution that minimizes the violation of soft constraints given below:

SC_1: Sum of the excesses of workers in each time slot of the planning horizon.
SC_2: Sum of the shortages of workers in each time slot of the planning horizon.
SC_3: Number of shifts employed.

The soft constraints have different importance depending from the problem. The objective function is the weighted sum of the three components, where the weights depend on the instance.

Notice that we consider designing shifts for a week and assume that the schedule is cyclic, i.e. the shifts that start on the last day of the planning period and stretch over midnight will have their end in the first day of the planning period. The cyclicity must be considered in applications where the night shifts interleave in some time slots with the morning shifts of the next day. Decomposing of problem in a daily basis would make the problem simpler to solve. However, if the days can have different staffing requirements during the week like in our case, the cyclicity of days can not be taken into consideration. Therefore, in such cases the time slots, where the night shifts interleave with the morning shifts, must be either covered only by night or by morning shifts, which may lead to worse solutions regarding the cover of staffing requirements.

3 Previous work

To the best of our knowledge the whole problem we consider in this paper has not yet been investigated in the literature. However, several related problems have been previously extensively studied. Dantzig developed the original set-covering formulation [7] for the *shift scheduling problem*, in which feasible shifts are enumerated based on possible shift starts, shift durations, breaks, and time windows for breaks. Aykin [2] introduced an implicit integer programming model for the shift scheduling problem in 1996 and later he compared an extended version [3] of his previous model with a similarly extended formulation introduced by Bechtold and Jacobs [4]. He observed Bechtold and Jacobs' approach needed fewer variables whereas his approach needed fewer constraints. Several problems were solved using both models with the integer programming software LINDO. The model proposed by Aykin was shown to be superior. Rekik et al. [17] developed two other implicit models and managed to improve upon previous approaches among them Aykin's original model. Tellier and White [19] developed a tabu search algorithm to solve a shift scheduling problem originating in contact centers which is integrated into the workforce scheduling system Contact Center Scheduling 5.5 from PrairieFyre Soft Inc. (`http://www.prairiefyre.com`). The solution of a shift scheduling problem with a planning period of one day, and at most three breaks (two 15 minutes breaks and a lunch break of 1 hour) has been considered in [6] and [16]. In [6] the authors make use of automata and context-free grammars to formulate constraints on sequences of decision variables. Quimper and Rousseau [16] investigate modeling of the regulations for the shift scheduling problem by using regular and context-free languages and solved the overall problem with Large Neighborhood Search. In addition to the

previous model, the authors applied their methods in single and multiple activity shift scheduling problems.

We note that our problem has several differences to the shift scheduling problems considered in the literature. We allow much larger number of breaks per shift (up to 10 breaks), the planning period is one week and we take into consideration the cyclicity. Moreover, our problem definition imposes some other constraints on shifts and breaks. Therefore, we can not make a direct comparison with the mentioned approaches in the literature.

Our problem can be divided in two sub-problems (shift design and break scheduling) which were considered separately in previous literature. Shift design problem regards generation of shifts without breaks for a week and has been considered in [13] and [9]. Musliu at al. [13] proposed a tabu search based algorithm to solve this problem. Additionally, their method orders moves and applies these first to regions with larger conflicts (larger over/under-staffing). The proposed solution methods have been used since several years in the commercial software package OPA of Ximes Inc. Di Gaspero et al. [9] proposed the application of hybrid approaches based on local search and min-cost max-flow techniques. The hybrid algorithm improved results reported in [13] on benchmark examples.

The break scheduling problem that imposes same constraints for breaks defined in this paper has been investigated in [5,20,14]. Beer et al. [5] applied a min-conflict based heuristic to solve this problem. This method has been applied in a real-life application for the supervision personnel. Results presented in [5] has been further imporved by memetic algorithms proposed in [14,20]. Note that a simplified break scheduling problem can be formulated as temporal constraint satisfaction problem (STP) [8] therefore it can be solved in polynomial time. This algorithm can be applied to find legal position of breaks, but without taking into consideration over-staffing and under-staffing that are very important criteria when solving shift design and break scheduling problems.

4 The Hybrid LS-CP Solver

We devise a hybrid strategy for tackling this problem that combines a Local Search (LS) method for determining the shifts with a Constraint Programming (CP) model for assigning breaks. In the following we outline some basic aspects of LS and CP, and we detail how we blend them in the hybrid solver.

4.1 LS and CP Basics

Local Search [11] is a family of methods to solve combinatorial optimization problems based on the definition of *proximity* (or *neighborhood*): a LS algorithm typically moves from a solution to a near one, trying to improve an objective function, iterating this process. LS algorithms generally focus the search only in specific areas of the search space, so they are *incomplete methods*, in the sense that they do not guarantee to find a feasible (or optimal) solution, but they search non-systematically until a specific stop criterion is satisfied.

In order to apply LS to the problem, we have to specify three main elements: the *search space* S, the *neighborhood* relation $\mathcal{N}(s)$ and the *cost function* f.

The search space consists of the set of potential solutions of a problem, expressed in accordance to a given problem model. A LS algorithm starts from an initial state s_0 (which can be obtained with some other technique or generated randomly) and enters a loop that *navigates* the search space, stepping from a state s_i to one of its neighbors $s_{i+1} \in \mathcal{N}(s_i)$. The search is controlled by a strategy driven by the *cost function* f, that estimates the quality of each state and, without loss of generality, it has to be minimized.

One of the most common LS techniques is *Hill Climbing*. This technique has many variants, however in all cases the basic idea is to perform only moves that improve or leave unchanged (i.e. *sideways* moves) the value of the cost function. The way a move is selected characterize the different Hill Climbing strategies. The so-called *Randomized Hill Climbing (RHC)*, which has been employed in this work, draws a random move at each step of the search and accepts it only if it is a non-worsening move. A widely adopted stop criterion for RHC is based on stagnation detection: the search is stopped after a number of non-improving moves have been drawn (also called *idle iterations*).

Constraint Programming [1] is a declarative programming paradigm, in which combinatorial optimization problems are encoded using a language consisting in variables and constraints and solved by a CP solver.

This paradigm is based on *complete methods* that analyze the search space alternating deterministic (constraint propagation, that is, the removal of values that cannot be assigned to any solution) and non-deterministic (variable assignment) phases. This process is iterated until a solution is found or unsatisfiability is reached; in the latter case the process backtracks to the last choice point (i.e., the last variable assignment) and tries other assignments.

In order to apply CP to a given problem it is necessary to model the problem in terms of variables and constraints that must hold. The solving procedure is then implemented in the CP solver and it resorts to some sort of tree search.

4.2 LS for Shift and Break Design

In the Shift and Break Design settings, LS deals with a search space composed of a set of shifts $\mathcal{S}$, each of them determined by its interval, and for each day of the time horizon the number of workers assigned and the size of the set of breaks. Notice that LS works on a partial representation of the solution, since the breaks are only specified in their number but not in the intervals they span. To complete this representation to a full solution we resort to a CP model (described below) whose purpose is to determine the interval variables of each break.

The set of shifts, $\mathcal{S}$, includes a number of *inactive* shifts, which have been assigned no worker on all days. These shifts are useful since, even though they do not contribute to the solution quality in their state, they can be possibly assigned workers during the search (i.e., become active) if their use leads to a solution improvement. Conversely, the shifts having at least a worker assigned in one day are called *active*.

Concerning the procedure for generating a random initial solution, we create a fixed number of random distinct active and inactive shifts for each shift type (for these instances a number of four active and two inactive shifts has proven to be adequate). Afterwards, for the active shifts, we assign a random number of employees for each day.

The neighborhood relations are similar to those considered in [9]. The moves have been modified to deal with the addition of the number of breaks. Moreover, a move dealing directly with the break component and a move for solution perturbation (which aims in decreasing the number of shifts) have been added. In detail the moves employed are the following:

- $\mathcal{N}_1$ ChangeStaff$(i, d, m \in \{\uparrow, \downarrow\})$: the staff of shift s_i on day d is increased ($\uparrow$) or decreased ($\downarrow$) by one employee.
- $\mathcal{N}_2$ ResizeShift$(i, m \in \{\uparrow, \downarrow\}, q \in \{\leftarrow, \rightarrow\})$: the length of shift s_i is increased or decreased by one timeslot, on the left- ($\leftarrow$) or on the right-hand ($\rightarrow$) side. This move is applied in a shift only if the modified shift does not violate hard constraints regarding its length and start.
- $\mathcal{N}_3$ ChangeBreaks$(i, d, m \in \{\uparrow, \downarrow\})$: the number of breaks of shift s_i on day d is increased or decreased by one.
- $\mathcal{N}_4$ MergeShifts(i, j): the two shifts s_i and s_j are merged together; the employees assigned to them are added and the interval of the resulting shift and the number of breaks for each day are averaged.

All the moves but ChangeStaff are meaningful on active shifts only, therefore they will be applied only on them. The ChangeStaff move, instead, is used also to change the state of a shift from active to inactive or vice versa.

The cost function is the weighted sum of the deviation (excess and shortage, with different weights) from the working requirements at each timeslot plus an additional weighted component that accounts for the number of active shifts employed in the solution. Notice that to allow an accurate computation of the deviation from the requirements a full solution is needed, therefore the cost function has to be computed only after a full solution has been determined by solving the CP model by means of a CP solver.

The neighborhood relations are intermingled in an Iterated Local Search (ILS) strategy [12], which exploits the set union of ChangeStaff, ResizeShift and ChangeBreaks neighborhoods —driven by RHC— followed by a solution perturbation obtained by applying a few random MergeShifts moves. The pseudocode of this procedure is shown in Fig. 3. The procedure is repeated as long as an overall timeout is not expired.

The Local Search procedure has been implemented using EasyLocal++ [10], and the software has been compiled with the GNU C++ compiler (v. 4.1.2).

4.3 The CP-Based Break Assignment

The breaks are determined by the solution of a CP model that considers all the shift variables (V_1)–(V_3) as fixed and searches for a suitable assignment of breaks

draw a random initial solution for the shift decision variables and store it in s_c
determine break assignment on s_c by means of the CP model
$s^* := s_c$
while ¬timeout expired **do**
  $\iota := 0$
  repeat
    / Apply RHC first */*
    $j :=$ RandomInt(1, 3)
    $m :=$ RandomMove($\mathcal{N}_j$)
    $s'_c := s_c \odot m$
    determine break assignment on (a subset of) s'_c by means of the CP model
    if $f(s'_c) < f(s_c)$ **then**
      $s_c := s'_c$
      $\iota := 0$
    else
      $\iota := \iota + 1$
    end if
  until $\iota > \max_\iota$
  if $f(s_c) < f(s^*)$ **then**
    $s^* := s_c$
  end if
  / Then perturb solution trying to merge k shift pairs */*
  $m :=$ RandomMove($\underbrace{\mathcal{N}_4 \times \ldots \times \mathcal{N}_4}_{k}$)
  $s_c := s_c \odot m$
end while

Fig. 3: The Iterated Local Search strategy for determining shift decision variables.

Input: the workforce requirements r_t, and a partial solution s, consisting in the determination of (active) shifts, workers, and number of breaks B_{id}
set up variables $\{\beta^b_{ide} | b = 1, \ldots, B_{id}, i = 1, \ldots, |\mathcal{S}|, d = 1, \ldots, D, e = 1, \ldots, w_{id}\}$
post break constraints on variables β^b_{ide}
if inconsistent state **then**
  return $\langle \emptyset, +\infty \rangle$
end if
optional: fix some of the β^b_{ide} variables and modify r_t accordingly
set free β^b_{ide} variables as branching variables
set variable f_{obj} as the weighted deviation from the (possibly modified) workforce requirements r'_t on all timeslots $t \in \mathcal{T}$
while ¬ timeout **do**
  branch and bound on β^b_{ide}, as driven by the values of f_{obj}
end while
if all β^b_{ide} variables are assigned **then**
  return $\langle \{\beta^b_{ide}\}, f_{obj} \rangle$
else
  return $\langle \emptyset, +\infty \rangle$
end if

Fig. 4: The CP-based procedure for determining the break decision variables.

(V_4) accordingly. More in detail, since the size of the breaks set on each day has been already determined by the LS solver, the decision variables considered in the CP break assignment model are only the `start`s and `length`s of the breaks, which are also subject to the set of constraints described in Section 2 (including the ones on the complementary *working periods* variables). Furthermore, the (weighted) deviation from the workforce requirements is computed for all the timeslots involved in the break assignment and it is used as the cost measure for the underlying Constrained Optimization Problem. The CP model has been coded in Gecode [18] and solved by the standard Branch & Bound procedure implemented in that system. The CP-based procedure is shown in Fig. 4.

It is worth noticing that this model can be easily applied also to break assignment subproblems that involve only a subset $\mathcal{S}' \subseteq \mathcal{S}$ of the shifts employed in the current solution. To this aim, it is sufficient to consider a set of modified workforce requirements r'_t, which can be obtained from r_t by subtracting all the employees working in the shifts $s \in \mathcal{S} \setminus \mathcal{S}'$ (line 7 in Fig. 4). Such an approach is similar to the Large Neighborhood Search (LNS) [15], in which a subset of the decision variables of the current solution are left free and the neighborhood induced by their possible combinations of values is explored by CP.

From Fig. 3, it is possible to observe that the CP model is employed in two stages of the Iterated Local Search procedure. At first (line 2), the model is used to assign breaks to a randomly generated initial solution. In this context, the whole set of shifts is passed to the CP-based procedure and the best solution computed by the procedure within a timeout of 25 seconds is retrieved. For most of the instances we tested, this time is enough to allow the CP solver to find a complete solution, however for a few instances the CP solver will not be able to find an initial solution. In the latter cases we overcome this problem by determining the break assignment of each shift taken in isolation. That is, we establish an order on the shifts and start determining the breaks for the first shift, then we fix these values and proceed with the second shift, further we fix the values of the first two shifts and proceed with the third shift, and so on.

The second use of the CP model (line 11 of Fig. 3) is in the determination of the breaks after performing a move. In this context, we decided to fix the break decision variables of all the shifts not involved in the move. That is, after a shift have been modified by a move all its breaks are optimized again by the CP-based procedure. Since in this case the size of the search space is much smaller than those of the whole problem, we let the CP solver to search for the best solution imposing a timeout of 0.5 seconds. In preliminary experiments we found out that this value allows to explore a satisfactory portion of the search tree.

5 Experimental Evaluation

To evaluate our algorithms we use existing benchmark instances in the literature for shift design and break scheduling. A set of real-life instances were provided from the authors of [5]. In addition, we apply our algorithms to randomly gen-

erated benchmarks. All instances are publicly available at `http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks`.

All instances have been solved at a time granularity of 5 minutes. In each instance are given staffing requirements for one week. The number of required employees in each time slot depends from the particular instance. For the real life instances up to 11 employees are required in one time slot, whereas for randomly generated instances the number of needed employees is up to around 27 for each time slot. The weight for shifts is 60, and the weights for shortage and excess are 10 and 2, respectively. These weights were selected based on experience with solving problems in real life applications.

Since, this is the first attempt to solve the whole shift design and break assignment problem, there are no previous results to compare with. Therefore, the aim of the experimental evaluation is to analyze the behavior of the hybrid algorithm on the basis of its components. We mainly evaluate the different combinations of *variable selection* heuristics in the CP framework. The purpose of these heuristics is to guide the CP search toward the most promising regions of the search tree by an early detection of failing assignments (according to the fail first principle). In this work we tested the following 6 strategies: *Random, Maximum Degree, Maximum Accumulated Failure Count, Minimum Size, Minimum Size/AFC, Minimum Size/Degree.*

Another design choice would have been the *value selection* heuristic for CP. However, in a preliminary experimental phase we found out that this choice was rather irrelevant. Therefore, in the following we apply always the selection of values to be assigned in increasing order. All the experiments were allowed 1 hour of computing time on an Intel QuadCore PC running Debian Linux 4.0.

In Fig. 5 we plot the progress of the hybrid algorithm equipped with different strategies for variable selection on two instances of the benchmark set (a real life and a randomly generated one). The qualitative behavior of the different strategies is consistent across all the instances and they show a slight dominance of the AFC_MAX heuristic w.r.t. the other heuristics tested.

For the purpose of investigating the reasons of the dominance of AFC_MAX, we analyze the exploration capabilities of the different variable selection heuristics. In Fig. 6 we plot the distribution of the improvements attained by CP during the exploration of the search tree. Data have been collected along the full executions of the algorithm on the whole benchmark set. The improvements are measured as the difference between the cost of the first and the last solution found by CP within the time granted. In the picture, the rightmost bar marked with a P, refers to the frequency of solutions directly found by applying constraint propagation only. Those solutions are quite valuable, since they are proven to be optimal without the need of additional tree search.

Is it possible to observe that, differently from other heuristics, AFC_MAX features a long left tail, leading to cost improvements up to 400 cost units. The superiority of AFC_MAX is also confirmed by the number of improving solutions explored by CP, reported in the last column of Table 1. In this table, we also compute other statistics for further analyzing the behavior of the heuristics in

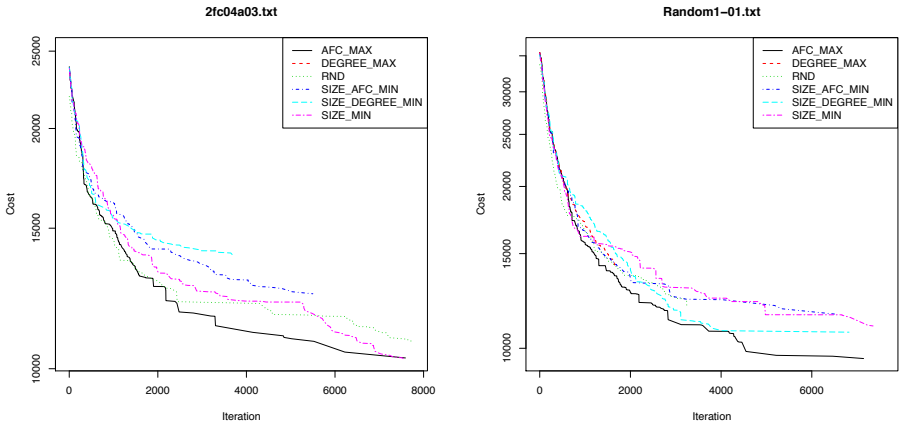


Fig. 5: Behavior of the algorithm on a real life and a random instance.

Variable Selection	$\rho(impr, \#nodes)$	$\rho(impr, \frac{\#nodes}{\#fail})$	$\overline{\#nodes}$	$\overline{\#fails}$	$\overline{\#sols}$
RND	-0.286***	-0.434***	9692	4822	4.13
AFC_MAX	-0.228***	-0.343***	7011	3490	5.53
DEGREE_MAX	-0.311***	-0.373***	8180	4075	4.83
SIZE_MIN	-0.374***	-0.448***	6682	3318	5.49
SIZE_AFC_MIN	-0.261***	-0.375***	9113	4535	5.35
SIZE_DEGREE_MIN	-0.232***	-0.365***	7263	3611	5.12

Table 1: Statistics on the improvement attained by CP.

relation to the improvements attained. The first two columns show the correlation between the improvement and the number of nodes explored, and the ratio between the number of nodes explored and the number of failures, respectively. We expected these two measures to be related to improvements, and actually we found out that all the correlations are statistically significant ($p < 0.01$). However, unfortunately, these measures cannot be used to discriminate between the different heuristics. The same applies to the remaining columns, that contain the averages of the number of nodes explored, the average number of failures and the average number of (improving) solutions found. Unfortunately they also do not show any regularity that allows to discriminate.

Finally, in Table 2 we report the best results found by the different versions of our algorithm on the benchmark instances.

6 Conclusions

We have investigated a real life shift design and break scheduling problem that arises in different areas of industry. In the literature design of shifts and break

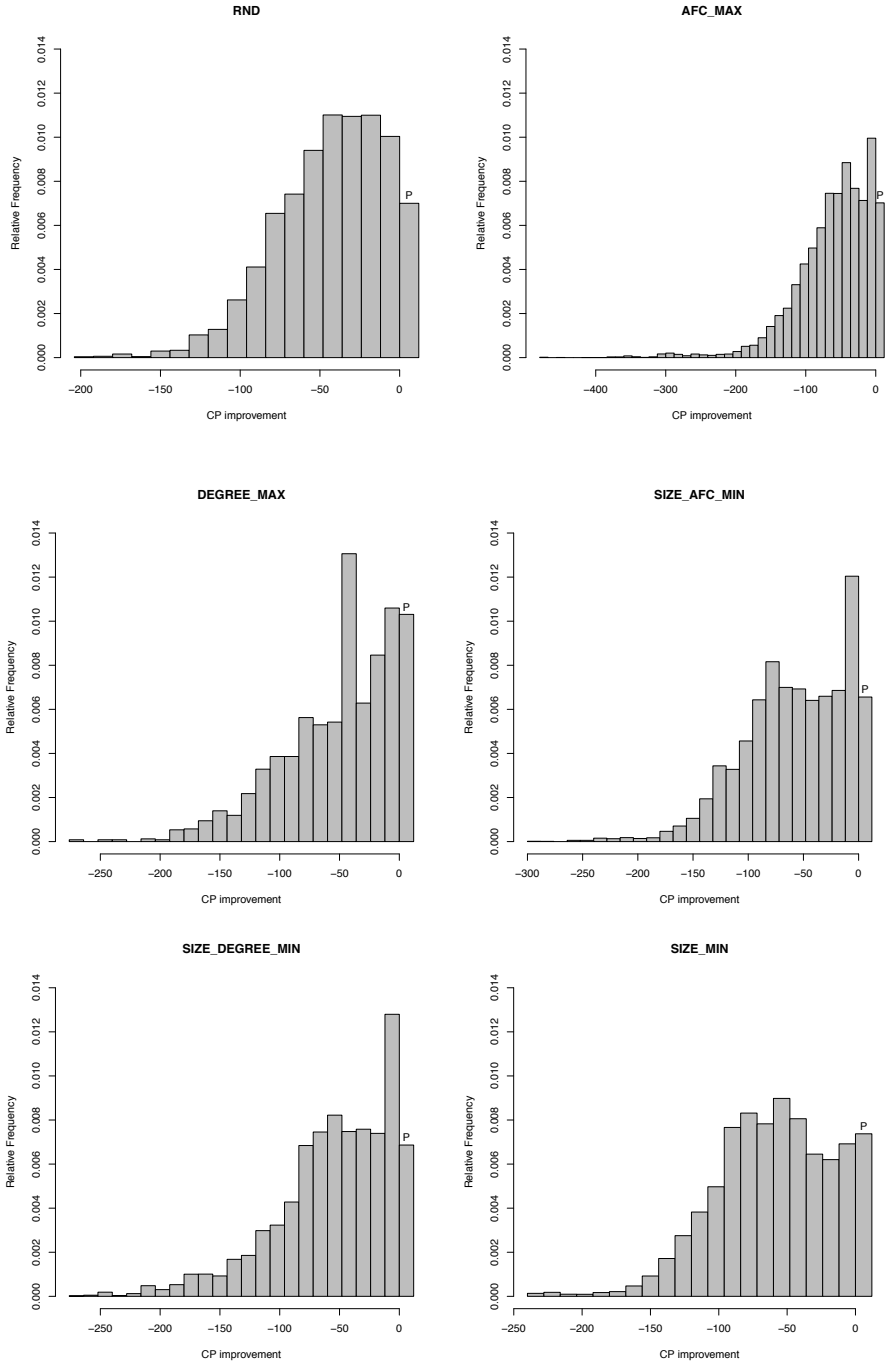


Fig. 6: Distribution of the improvements attained by CP on break assignment.

Instance	Shortage	Excess	# Shifts
2fc04a03	173	2636	23
2fc04a	182	2750	25
2fc04b	181	2892	21
3fc04a03	733	3168	20
3fc04a04	130	2732	21
3fc04a	256	3084	24
3si2ji2	137	2909	24
4fc04a03	94	2710	21
4fc04a	526	3064	19
4fc04b	200	2690	24
50fc04a03	175	2809	26
50fc04a04	180	2636	29
50fc04a	169	3150	20
50fc04b	200	2730	30
51fc04a03	410	2653	25
51fc04a04	209	2890	23
51fc04a	189	3051	25
51fc04b	250	3046	23
Random1-01	215	3355	28
Random1-02	405	4557	29
Random1-03	369	4800	34
Random1-04	421	5901	44
Random1-05	285	3713	23
Random1-06	246	2638	17
Random1-07	227	3046	28
Random1-08	330	4465	33
Random1-09	276	3930	25
Random1-10	389	4997	32
Random1-12	164	2850	17
Random1-13	303	3888	25
Random1-14	380	5763	47
Random1-15	179	1282	7
Random1-16	426	4610	39
Random1-17	373	5746	53
Random1-18	530	4724	32
Random1-19	368	3961	33
Random1-20	438	5247	32
Random1-21	242	3164	24

Instance	Shortage	Excess	# Shifts
Random1-22	186	3288	23
Random1-23	265	4468	34
Random1-24	330	3585	21
Random1-25	452	5608	45
Random1-26	698	3723	31
Random1-27	718	4602	35
Random1-28	258	3086	20
Random1-29	336	3973	39
Random1-30	250	2512	18
Random2-01	375	4376	25
Random2-02	289	3968	34
Random2-03	288	4319	39
Random2-04	311	3935	30
Random2-05	260	4161	34
Random2-06	330	5197	42
Random2-07	427	5187	40
Random2-08	323	4357	29
Random2-09	391	4461	35
Random2-10	432	5566	42
Random2-12	365	4929	42
Random2-13	421	5148	49
Random2-14	410	5678	53
Random2-15	518	5696	41
Random2-16	350	6255	49
Random2-17	452	4537	36
Random2-18	557	5413	37
Random2-19	604	5418	43
Random2-20	475	5769	40
Random2-21	371	6150	48
Random2-22	430	6312	48
Random2-23	402	5288	44
Random2-24	427	5634	44
Random2-25	487	5754	47
Random2-26	517	5836	47
Random2-27	529	5835	54
Random2-28	340	5157	45
Random2-29	506	5407	45
Random2-30	678	7041	60

Table 2: Best results found by the hybrid algorithms on the benchmark instances.

assignment have been solved in two separated phases, because of their high complexity. This required the application of an iterative process that usually could be performed only by the staff scheduling experts. To automate the solution of the whole problem and to minimize the need for domain experts, we proposed an innovative hybrid method that combines features of LS and CP techniques.

This paper investigates for the first time the solution of the complete shift design and break scheduling problem and the application of CP to the break assignment subproblem. The CP model has shown to be very practical for the local search to find legal break assignments that optimize over/under staffing. Our solver has been applied successfully on the existing real life and random instances in the literature. Obtained solutions fulfill all hard constraints and the solver gives promising results considering the level of fulfillment of soft constraints.

For the future, we plan to further study the hybridization of the LS and CP paradigms. In particular, we plan to investigate application of CP techniques for design of shifts and local search for break scheduling within the hybrid algorithm.

Acknowledgments. This research is partially conducted within the competence network Softnet Austria (`http://www.soft-net.at/`) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

References

1. K. R. Apt. *Principles of Constraint Programming.* Cambridge University Press, Cambridge (UK), 2003.
2. T. Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42(4):591–602, 1996.
3. T. Aykin. A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European J of Operational Research*, 125:381–397, 2000.
4. S.E. Bechtold and L.W. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990.
5. Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An AI-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25(2):60–73, 2010.
6. M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 2009.
7. G.B. Dantzig. A comment on eddie's traffic delays at toll booths. *Operations Research*, 2:339–341, 1954.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.
9. L. Di Gaspero, J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf, and W. Slany. The minimum shift design problem. *Annals of Operations Research*, 155:79–105, 2007.
10. L. Di Gaspero and A. Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software — Practice & Experience*, 33 (8):733–765, 2003.

11. H.H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA (USA), 2005.
12. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In G. Glover F., Kochenberger, editor, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell (MA), USA, 2002.
13. N. Musliu, A. Schaerf, and W. Slany. Local search for shift design. *European J of Operational Research*, 153(1):51–64, 2004.
14. N. Musliu, W. Schafhauser, and M. Widl. A memetic algorithm for a break scheduling problem. In *Proceedings of the VIII Metaheuristic International Conference (MIC 2009)*, Hamburg (Germany), July, 13-16 2009.
15. L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In M. Wallace, editor, *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP 2004)*, number 3258 in Lecture Notes in Computer Science, pages 468–481, Springer Verlag, Germany, 2004.
16. C.-G. Quimper and L.-M. Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *J of Heuristics*, 16(3):373–391, 2010.
17. M. Rekik, J-F. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *J of Scheduling*, 13(1):49–75, 2010.
18. Gecode Team. Gecode: Generic constraint development environment. `http://www.gecode.org`.
19. P. Tellier and G. White. Generating personnel schedules in an industrial setting using a tabu search algorithm. In E. K. Burke and H. Rudová, editors, *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 293–302, 2006.
20. M. Widl, N. Musliu. An improved memetic algorithm for break scheduling. In *Proceedings of the 7th International Workshop on Hybrid Metaheuristics (HM 2010)*. Lecture Notes in Computer Science, 2010.