Scheduling Breaks in Shift Plans for Call Centers

Andreas Beer · Johannes Gärtner · Nysret Musliu · Werner Schafhauser · Wolfgang Slany

Abstract In this paper we consider a real-life break scheduling problem for call center agents involving a large number of breaks and constraints. Obtaining good solutions for this problem increases the well-being of call center employees and guarantees a certain quality of service for calling customers. To solve this problem we present two local search approaches, a min-conflicts based search algorithm and a tabu search algorithm and consider a hybridization of both techniques. Our computational experiments reveal that the presented techniques generate high-quality solutions to our problems in reasonable time. These solutions are able to satisfy the needs of customers, call center agents, and employers at the same time.

Keywords Break scheduling · Min-conflicts heuristic · Tabu search · Call centers

1 Introduction

Break scheduling represents an important task, particularly relevant for working areas in which employees spent their entire working time in front of computer monitors, e.g., in call centers or airports. For people working under these conditions it is imperative to have a break from time to times, in order to reduce stress and to avoid exhaustion. Usually in each industry there exist labor rules regulating the quantity of break time for employees and specifying how these breaks must be scheduled in a legal shift

A. Beer · W. Slany
Inst. for Software Technology, Graz University of Technology, 8010 Graz, Austria
E-mail: abeer@ist.tugraz.at

W. Slany
E-mail: wsi@ist.tugraz.at

J. Gärtner
Ximes Inc, Hollandstraße 12/12, 1020 Vienna, Austria
E-mail: gaertner@ximes.com

N. Musliu · W.Schafhauser (✉)
Inst. for Information Systems, Vienna University of Technology, 1040 Vienna, Austria
E-mail: musliu@dbai.tuwien.ac.at

W. Schafhauser
E-mail: schafha@dbai.tuwien.ac.at

plan. Consequently, the great challenge of break scheduling problems is to create shift plans which satisfy the legal requirements on break quantities and break patterns and guarantee that a certain number of employees is present at any time.

In this paper we address a complex real-life break scheduling problem originating from a call center, which we call the *break scheduling problem*. As an input we are given a *shift plan*, the *break types* and *break quantities* to be scheduled for each shift and the *staffing requirements*, i.e., the number of working employees required during a certain time period. The specified break quantities have to be scheduled within the given shifts in such a way that the number of violations of various constraints on breaks and on the excess or shortage of working employees is minimized. The break scheduling problem may be regarded as an instance of a constraint satisfaction problem which is known to be NP-hard in general.

Local search techniques represent a promising approach to obtain solutions of good quality for complex optimization problems. In each step of their search they regard solutions closely related to the current one, the so-called local *neighborhood*, and select a particular solution from that neighborhood for further improvement. Usually, this neighborhood is computed by applying small changes to the current solution. In terms of local search techniques these small changes are denoted as *moves*.

Canon as well as Tellier and White considered shift design problems including the scheduling of breaks (Tellier and White 2006; Canon 2007). They were able to obtain solutions of acceptable quality for their respective problems by using local search techniques. In contrast to these related problems the break scheduling problem addressed in this paper consists in exclusively scheduling breaks within a fixed shift plan whereas in (Tellier and White 2006) and (Canon 2007) also shifts are a subject to the optimization process. Moreover, in the break scheduling problem we have to schedule large numbers of breaks per shift and consider various constraints not imposed on previous problems.

Motivated by the encouraging results of local search techniques for related problems we investigate local search algorithms for the break scheduling problem. At this point we want to state the main contributions of this paper:

- ▷ We propose and implement a min-conflicts based heuristic for the break scheduling problem and combine it with a random-walk strategy in order to escape from local optima.
- ▷ We develop and implement a tabu search algorithm for the break scheduling problem. Moreover we propose several variants of tabu lists to ensure that the tabu search does not get stuck in cycles.
- ▷ In addition, we consider a hybridization of both methods, a min-conflicts search using tabu lists.
- ▷ To evaluate our proposed methods we carry out comprehensive experiments on 23 real-life instances from a call center, which are presented in this paper for the first time. In our experiments we determine which parameters are best for various search strategies and apply the best performing search strategies to all instances. Our computational results reveal that in most cases min-conflicts search achieves the best results.

2 Problem Description

In this section we give a formal description of the break scheduling problem. In the break scheduling problem we regard shift plans for call center agents in which each shift must contain a certain quantity of breaks. Our goal is to schedule the breaks within the shifts in such a way that we obtain a solution of high quality according to various criteria. Formally, as input for the break scheduling problem we are given:

- ▷ a *planning period* formed by T consecutive time slots $[a_1, a_2), [a_2, a_3), ..., [a_T, a_{T+1})$ all having the same length *slotlength* (in minutes). Time points a_1 and a_{T+1} represent the beginning and end of the planning period. All time points have the same format *day:hour:minute*.
- ▷ n *shifts* $s_1, s_2, ..., s_n$ representing employees working within the planning period. Each shift s_i has the adjoined parameters, $s_i.start$ and $s_i.length$ representing its start and its length. Each shift corresponds to exactly one employee.
- ▷ *break quantities* and *break types* to be scheduled for each shift. We distinguish between two different types of breaks: lunch breaks and monitor breaks. The parameter $s_i.lunch$ stores the length of a shift's lunch break in minutes, whereas the parameter $s_i.monitor$ specifies a shift's monitor break quantity.
- ▷ the *staffing requirements* for the planning period. Each time slot $[a_t, a_{t+1})$ has an adjoined integer value w_t indicating the optimal number of employees that should be working during time slot $[a_t, a_{t+1})$. An employee is considered to be *working* during time slot $[a_t, a_{t+1})$ if in its corresponding shift no break is scheduled during time slot $[a_t, a_{t+1})$.

A *break* b is characterized by the parameters, $b.shift$ specifying its associated shift, its start $b.start$ and its length $b.length$. We assume that all parameters representing time points coincide with a time point a_t defining the start or the end of a time slot of the planning period. Moreover we expect each parameter representing a time length or a break quantity to be a multiple of *slotlength*.

Given a planning period, a set of shifts, the associated break quantities, and the staffing requirements, a *feasible solution* to the break scheduling problem is a set of breaks such that:

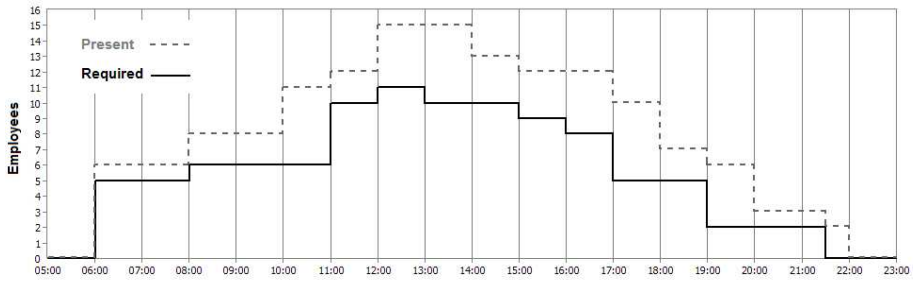
1. Each break lies entirely within its associated shift.
2. Two distinct breaks associated with the same shift do not overlap in time.
3. For each shift the sum of all its associated break lengths is exactly the specified break quantity for the shift, which is $s_i.lunch + s_i.monitor$.
4. If $s_i.lunch > 0$ then there is one break in s_i whose length is at least $s_i.lunch$.

Example 1 Figure 1 shows the staffing requirements (solid curve) for a real-life instance of the break scheduling problem. The curve of present employees (dashed curve) results from the given shift plan, which does not contain any breaks yet.

2.1 Constraints on Feasible Solutions

Among all feasible solutions for the break scheduling problem we aim at finding an optimal one according to various criteria. These criteria are modeled as constraints on feasible solutions. Basically we distinguish between four main groups of constraints, namely constraints on:

Fig. 1 Curves of required and present employees resulting from an instance of the break scheduling problem.



1. The position of breaks within shifts.
2. The length of breaks.
3. The distances between breaks.
4. The excesses and shortages of working employees according to staffing requirements.

2.1.1 Constraints on the Position of Breaks within Shifts

C_1 *MinimumDistanceToShiftBegin:* Each break may start not earlier than a given number of minutes after the beginning of its associated shift.
C_2 *MinimumDistanceToShiftEnd:* Each break must end not later than a given number of minutes before the end of its associated shift.
C_3 *MaximumDistanceToShiftBegin:* The earliest break of a shift must not start later than a given number of minutes after the beginning of the shift.
C_4 *MaximumDistanceToShiftEnd:* The latest break of a shift must not end earlier than a given number of minutes before the end of the shift.

2.1.2 Constraints on the Distances Between Breaks

C_5 *MinimumDistanceBetweenBreaks:* The temporal distance between two consecutive breaks must be at least a given minimum number of minutes.
C_6 *MaximumDistanceBetweenBreaks:* The temporal distance between two consecutive breaks must not exceed a given maximum number of minutes.

2.1.3 Constraints on the Length of Breaks

C_7 *MinimumBreakLength:* The length of each break must be at least a given minimum number of minutes.
C_8 *MaximumBreakLength:* The length of each break must not exceed a given maximum number of minutes.
C_9 *OptimumBreakLength:* The length of each break should be equal to a given optimum number of minutes.
C_{10} *MinimumLengthAfterDistance:* If the distance between two consecutive breaks reaches or exceeds a certain number of minutes the length of the latter break must be at least of a given minimum length.

2.1.4 Constraints on the Excess and Shortage of Working Employees

C_{11} *NoExcess:* In each time slot $[a_t, a_{t+1})$ the number of working employees, i.e., the employees who are not assigned a break in that time slot, should not exceed w_t.

C_{12} *NoShortage:* In each time slot $[a_t, a_{t+1})$ the number of working employees should be at least w_t.

C_{SD} *NoSquaredDeviation:* In many practical instances for the break scheduling problem the staffing requirements are significantly higher or lower than the number of scheduled employees during the overall planning period. Consequently, each solution will always produce the same amount of excess or shortage. For such instances we introduced an additional constraint aimed at minimizing the squared deviation from staffing requirements in each time slot. Informally speaking, this constraint prefers solutions whose curve of working employees has a shape similar to the curve representing the staffing requirements.

2.2 Extending the Problem with Breaks of Fixed Length

When scheduling breaks within shift plans it is sometimes necessary to constrain a single break differently from the remaining breaks within its shift. For instance, employees prefer to have a one hour lunch break at the middle of their duty or between 11:00 and 14:00. Therefore we introduce a constraint defined on a single break within a shift.

C_{13} *FixedBreak:* Each shift can contain a break of a certain specified length, which may differ from the lengths required by other constraints. Optionally, this break must lie within some given *allowed time range*, preferably within a given *optimum time range*. The break must not be scheduled within a given *forbidden time range*.

Note: The criteria required by the constraint *FixedBreak* may contradict the requirements of several previously introduced constraints. For that reason, the following constraints are not applied to that single break of desired length: *MinimumDistanceToShiftBegin* (C_1), *MinimumDistanceToShiftEnd* (C_2), *MinimumBreakLength* (C_7), *MaximumBreakLength* (C_8), and *OptimumBreakLength* (C_9).

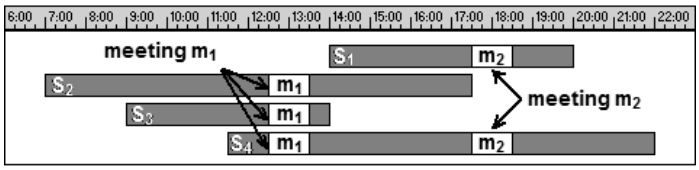
2.3 Extending the Problem with Meetings

Call center employees can take part in meetings during their working time. While attending meetings call center agents do not process incoming phone calls. Thus, during the time a meeting takes place the participating employees are not considered to be working with respect to staffing requirements.

Example 2 In the shift plan given in Figure 2 call center employees represented by shifts s_2, s_3, and s_4 take part in meeting m_1 taking place from 12:30 until 13:30. Employees working in shifts s_1 and s_4 attend meeting m_2 from 17:30 to 18:30.

In order to handle meetings we have to extend the break scheduling problem further. Moreover we introduce an additional constraint concerning the break time scheduled in meetings. In addition to the input for our basic problem we are given:

Fig. 2 A shift plan containing meetings.



- ▷ k meetings $m_1, m_2, ..., m_k$. Each meeting m_j has two adjoined parameters $m_j.start$ and $m_j.length$, specifying its start time and its length. Moreover, each meeting has an adjoined set $m_j.S$. Set $m_j.S$ contains shifts and indicates that employees assigned to these shifts participate in meeting m_j. Additionally we are given an integer value q_j specifying the break time required to be scheduled during meeting m_j.

C_{14} *BreakQuantityInMeeting:* For each employee participating in a meeting m_j we require that exactly q_j minutes of break time are scheduled within meeting m_j.

Meetings have the following side effects on several constraints of the basic break scheduling problem:

C_5, C_6, C_{10}: These constraints are only relevant for breaks not scheduled during the same meeting. In other words, these constraints are ignored for consecutive breaks ending and starting during the same meeting.

C_7, C_8, C_9: We consider only those parts of breaks which are scheduled outside the time range of a meeting, disregard breaks of a certain fixed length, and refer to these breaks as *breaks outside a meeting*. The constraints on the minimum, maximum, and optimum break length are only applied to these breaks outside a meeting.

C_{11}, C_{12}, C_{SD}: While participating in meetings employees are not considered to be working. Breaks scheduled during meetings do not further decrease the number of working employees.

3 Local Search Techniques for the Break Scheduling Problem

Encouraged by the results produced by local search techniques on related problems (Canon 2007; Tellier and White 2006; Gaspero et al. 2007; Musliu et al. 2004), we decided to develop several local search algorithms for the break scheduling problem, namely a tabu search as well as a min-conflicts search based approach.

3.1 Principles of Local Search Techniques

Local search techniques try to obtain a good solution for an optimization problem by iteratively improving an initial solution.

To design a local search algorithm for a specific optimization problem, in particular for the break scheduling problem, we have to develop the following key ingredients of a local search algorithm:

- A representation of solutions of the specific optimization problem.
- An objective function mapping solutions to real values. The value of a solution acts as a measure for its quality.
- Moves applicable to a solution in order to compute its neighborhood.

3.2 Representation of Solutions for the Break Scheduling Problem

We represent the solution for the break scheduling problem as a set of breaks. Each break has a variable start and constant length. Moreover, each break is associated with a certain shift and must lie entirely within that shift's range. Given a shift and its quantities of lunch and monitor breaks we distribute the break time among the following three types of breaks:

fixed breaks: For each break required by constraint *FixedBreak* (C_{13}) we generate a so-called fixed break having the desired length. If possible the entire lunch break quantity is part of a single fixed break.

lunch break: If it is not possible to schedule the lunch break within a fixed break we generate a lunch break. Each shift may contain at most one lunch break comprising its total lunch time quantity.

monitor breaks: The remaining time not planned as fixed breaks and lunch breaks is scheduled within monitor breaks. We try to assign a monitor break the optimal break length as required by constraint *OptimalBreakLength* (C_9) to each monitor break but the last monitor break may be shorter than the desired optimum length.

The obtained breaks are scheduled randomly in their respective shifts such that the the obtained solution is feasible and satisfies the constraints $MinimumDistanceToShiftBegin(C_1)$ and $MinimumDistanceToShiftEnd(C_2)$. This solution acts as the initial solution for our proposed local search techniques.

3.3 Objective Function

The break scheduling problem can be modeled as a multi-criteria optimization problem where an objective function is to be minimized. The importance of a single criterion and the corresponding constraint varies from task to task. Thus, the break scheduling problem's objective function can be modeled as a weighted sum of the violation degree of each constraint, or more formally:

$$F(Solution) = \sum_{i=1}^{14} W_i \cdot violations(C_i) + \frac{violations(C_{SD})}{2 \cdot ub(C_{SD})}$$

In the objective function presented above, $ub(C_{SD})$ denotes an upper bound on the violation degree of the constraint *NoSquaredDeviation*. If two solutions have the same objective value according to constraints $C_1, ..., C_{14}$ the objective function prefers the solution with a smaller squared deviation from staffing requirements.

3.4 Moves and Local Neighborhood

On the basis of the representation of a solution for the break scheduling problem we developed two kinds of moves.

assignment: a break is moved to a new position within its associated shift, by assigning it a new start.

swap: the starts of two breaks associated with the same shift are exchanged, meaning that the breaks are actually swapped.

Given a feasible solution S to the break scheduling problem we define its neighborhood $N(S)$ to be the set of all solutions obtained by applying an assignment on a single break in S or by swapping two breaks within the same shift in S. As a *legal move* we consider any assignment or swap guaranteeing that after the move:

1. The breaks in the affected shifts are not overlapping.
2. The lunch break is not scheduled in a meeting.
3. The affected breaks lie within their allowed time regions specified by the constraints *MinimumDistanceToShiftBegin* (C_1), *MinimumDistanceToShiftEnd* (C_2) and *FixedBreak* (C_{13}).
4. Fixed breaks are not preceded or succeeded by any other break after the move.

3.5 Tabu Search

A local search method may loop over the same set of solutions again and again. In that case the search is not able to discover better solutions than those already considered. In order to overcome this problem Glover proposed a local search technique named tabu search (Glover and Laguna 1997). The main idea behind tabu search is to maintain a "memory" denoted *tabu list* which stores recently visited solutions and forces the search to explore new regions of the search space. At this point we would like to give a brief overview on the different kinds of tabu lists:

tabu solutions: This is the most basic form of tabu search: We are given a fixed number – the *tabu length* (`tbl`) – and remember the last `tbl` solutions encountered. These are tabu, meaning that even if they are part of the current neighborhood, moving there will not be considered.

tabu attribute: In some applications it is sufficient to consider certain attributes of a solution tabu: Any solution having one of these attributes will not be considered for evaluation for `tbl` iterations.

tabu move: Instead of setting solutions or their attributes tabu, we prohibit just a certain move for the immediate future.

variable tabu length: The number of iterations during which a solution, an attribute or a move is considered to be tabu does not need to be a fixed value. It can increase and decrease during the search. Small sizes of the tabu list lead to an intensification of the search on the region around the current solution whereas great sizes induce a diversification of the search, meaning that the search is forced to explore new regions of the search space.

There is one important note that needs to be made concerning the tabu status: If a solution is both tabu and better than the best solution found so far it will be wise to disregard its tabu status. This behavior is called "aspiration".

With the more general groundwork taken care of, we now want to look at the tabu options that were implemented in our tabu search. Note that all options include aspiration criteria. For the break scheduling problem we have experimented with two kinds of tabu lists, namely tabu lists containing:

breaks: After performing an assignment or swap, the affected breaks are added to the tabu list. For the next `tbl` iterations, they are considered tabu.

inverse moves: After performing a move we add its inverse move to the tabu list. After assigning a break a new position we prohibit the reassignment of its old position for `tbl` iterations, after swapping two breaks we do not allow to swap these breaks again for `tbl` iterations.

All variants of tabu lists described above are available either with fixed or with variable tabu length. In the latter case `tbl` varies between a (parameterizable) minimum and maximum length. If a move improves the current solution, the tabu length is decreased by one, otherwise it is increased by one.

3.6 Min-Conflicts Search

Minton et al. present another local search strategy named *min-conflicts search* which has been used successfully for solving several constraint satisfaction problems (Minton et al. 1992). Unlike tabu search the min-conflicts search strategy

- explores a smaller neighborhood in each step. In fact only the neighborhood of a single variable of an constraint satisfaction problem is considered during a single iteration.
- concentrates on variables causing constraint violations.

Motivated by the work in (Minton et al. 1992) we adapted the min-conflicts search for the break scheduling problem. The search starts with an initial feasible solution for the break scheduling problem and tries to improve it incrementally. In each iteration the min-conflicts heuristic

- selects a break violating a constraint.
- determines a move minimizing or at least not worsening the violation degree of the current solution.

If such a move exists it will be applied to the current solution and the search continues until some halting condition is satisfied. The proposed min-conflicts search applies only moves which do not decrease the quality of the current solution. Thus, once the search has reached a *local optimum*, that is, a solution better than any solution within its neighborhood, it will not be able to escape it. At that point the algorithm does not explore further regions of the search space which might contain solutions of better quality than those we have considered so far.

To avoid this undesired behavior we adapted a random walk strategy for satisfiability problems (Selman et al. 1993). With a certain probability p we perform an arbitrary move to a break violating a constraint. On the one hand the violation degree of the resulting solution may be worse than the previous one but on the other hand performing such moves may help us to overcome local optima. The ordinary min-conflicts search is carried out with the remaining probability $1 - p$. The resulting algorithm is called Min-Conflicts-Random-Walk and is presented in Algorithm 1 in pseudo-code notation. As halting criterion we use a specified number of seconds.

Algorithm 1: Min-Conflicts-Random-Walk for the break scheduling problem

Input : An instance of the break scheduling problem.
The random walk probability p.
Result: A feasible solution for the break scheduling problem.

Generate an initial solution S.
repeat

`/* Min-Conflicts Search */`
With probability $1-p$ **do**
Randomly select a break $b \in S$ causing a constraint violation.
Apply the best legal move for b not increasing $Violations(S)$; results in new S.
Ties are broken randomly.
end

`/* Random Walk */`
With probability p **do**
Randomly select a break $b \in S$ causing a constraint violation.
Apply an arbitrary legal move for b; results in new S.
end

until $F(S) = 0$ *or* $halting\_criterion = true$

return *the best solution found during the search.*

3.6.1 Extending Min-Conflicts Search with a Tabu List

To avoid cycles in the search we additionally applied tabu mechanisms in the algorithm min-conflicts-random-walk. During the min-conflicts part of the algorithm we apply only moves which are not tabu according to a tabu list. As tabu lists we used the two approaches presented in section 3.5.

4 Computational Results

In this section we evaluate our proposed algorithms for the break scheduling problem on 23 real-life benchmark instances from a call center. We conducted several experiments to investigate the performance of the different local search techniques and to answer the following questions:

1. Does the random-walk strategy improve the results returned by the min-conflicts search algorithm?
2. Which tabu list performs best on the considered benchmark instances?
3. Which of the proposed methods returns the best results?
4. What is the quality of the solutions returned by the considered algorithms?

All our experiments were conducted on the following two machines: a Genuine Intel(R) T2400 1.83 GHz processor having 2 GB RAM and an Intel(R) Core(2) Duo T7300 1.99 GHz processor having 3 GB RAM. All experiments concerning min-conflicts search and its combinations with a tabu list were carried out on the first computer whereas the experiments on tabu search techniques were run on the second machine. Since our local search algorithms include randomization we carried out ten runs for each instance in each experiment. A single run was executed with a ten minute time limit.

To implement our local search techniques and to model constraints we used the programming language COMET (Hentenryck and Michel 2005). The programming language COMET provides an expressive constraint language to model combinatorial optimization problems. Moreover, COMET also offers a rich search language that abstracts different components of local search algorithms.

4.1 Benchmark Instances for the Break Scheduling Problem

To evaluate our proposed methods we used 23 real-life examples for the break scheduling problem originating from a call center. Table 1 lists these instances and presents their basic properties like the number of shifts (column Shifts), the length of the planning period (column Period) and the length of each time slot in the respective problem (column Slot). Column Breaks contains the total number of breaks for each instance obtained as described in Section 3.2. For all instances, the lunch and monitor break quantities are dependent on the length of a shift. For a single shift s_i the lunch break and monitor break time are computed as follows:

$$s_i.lunch = \begin{cases} 30 \text{ minutes, if } s_i.length > 360 \text{ minutes} \\ 0 \text{ minutes, else} \end{cases}$$

$$s_i.monitor = \lfloor (s_i.length - s_i.lunch)/60 \rfloor \cdot 10 \text{ minutes}$$

We distinguish between three types of instances, Basic, Fixed and Meeting. Basic_01-Basic_12, represent instances for the basic break scheduling problem omitting breaks of fixed length and meetings. Fixed_01-Fixed_07, require additionally a fixed one-hour break in each shift, which is imposed using constraint *FixedBreak* (C_{13}) from Section 2.2. The remaining four instances, Meeting_01-Meeting_04, additionally contain meetings.

The three types of instances differ also in the specified limits on the position of breaks, distances between breaks and break lengths. Table 2 describes the different temporal limits imposed by constraints for each type of instance. For instance, from the first row of Table 2 we see that for instances of type Basic the distance of a break to the beginning of its shift must be at least an hour.

In instances Fixed_01-Fixed_07 a further constraint *FixedBreak* (C_{13}) is imposed regulating the allowed, optimal, and forbidden time range for the required one-hour break. These ranges are given in Table 3. The allowed time range for the single break is specified by a required minimum distance to the beginning and end of a shift. The optimum and forbidden time ranges specify the time period in which the break should and must not be located respectively.

Finally, Table 4 presents the weight of each constraint for the given real-life tasks.

4.2 Impact of the Random-Walk Strategy

In the first experiment we investigated whether the random-walk strategy has an impact on the quality of solutions returned by the min-conflicts-search algorithm. In order to answer that question we ran our min-conflicts search on ten instances, with different random-walk probabilities of 0%, 1%, 2.5%, 5% and 10%. Based on the average quality

Table 1 Real-life benchmark instances for the break scheduling problem.

Instance	Shifts	Period	Slot	Breaks
Basic_01	24	15:00	00:05	76
Basic_02	24	15:00	00:05	76
Basic_03	24	15:00	00:05	76
Basic_04	24	15:00	00:05	82
Basic_05	26	15:00	00:05	85
Basic_06	32	15:00	00:05	93
Basic_07	28	15:00	00:05	94
Basic_08	28	15:00	00:05	108
Basic_09	28	15:00	00:05	110
Basic_10	15	16:00	00:10	55
Basic_11	15	15:30	00:10	56
Basic_12	20	16:00	00:10	67
Fixed_01	15	09:00	00:05	45
Fixed_02	15	16:00	00:10	41
Fixed_03	15	16:00	00:10	41
Fixed_04	15	16:00	00:10	41
Fixed_05	15	16:00	00:10	41
Fixed_06	15	15:30	00:10	45
Fixed_07	20	16:00	00:10	53
Meeting_01	14	12:00	00:05	36
Meeting_02	24	15:00	00:05	76
Meeting_03	25	15:00	00:05	87
Meeting_04	27	15:00	00:05	96

Table 2 Temporal limits imposed by constraints on the different types of instances.

	Constraint	Basic	Fixed	Meeting
C_1	$MinimumDistanceToShiftBegin$	01:00	01:00	01:00
C_2	$MinimumDistanceToShiftEnd$	01:00	01:00	01:00
C_3	$MaximumDistanceToShiftBegin$	02:30	03:00	02:30
C_4	$MaximumDistanceToShiftEnd$	02:30	03:00	02:30
C_5	$MinimumDistanceBetweenBreaks$	01:00	01:00	01:00
C_6	$MaximumDistanceBetweenBreaks$	02:30	03:00	02:30
C_7	$MinimumBreakLength$	00:10	00:10	00:10
C_8	$MaximumBreakLength$	01:00	01:00	01:00
C_9	$OptimumBreakLength$	00:30	00:20	00:30
C_{10}	$MinimumLengthAfterDistance$	00:30	00:30	00:30

Table 3 The allowed, optimal and forbidden time range for one-hour breaks in instances Fixed_01-Fixed_07.

	Min. Dist. to		Optimal	Forbidden
Instance Name	Begin	End	Range	Range
Fixed_01	02:00	00:00	11:30 - 14:30	15:00 - 18:00
Fixed_02	02:00	01:00	11:30 - 14:30	15:00 - 18:00
Fixed_03	02:00	01:00	11:30 - 14:30	15:00 - 18:00
Fixed_04	02:00	01:00	11:30 - 14:30	-
Fixed_05	02:00	01:00	-	-
Fixed_06	02:00	01:00	11:30 - 14:30	15:00 - 18:00
Fixed_07	02:00	01:00	11:30 - 14:30	15:00 - 18:00

Table 4 Weights of constraints for the considered real-life instances.

	Constraint	Weight W_i
C_1	$MinimumDistanceToShiftBegin$	10
C_2	$MinimumDistanceToShiftEnd$	10
C_3	$MaximumDistanceToShiftBegin$	100
C_4	$MaximumDistanceToShiftEnd$	100
C_5	$MinimumDistanceBetweenBreaks$	10
C_6	$MaximumDistanceBetweenBreaks$	100
C_7	$MinimumBreakLength$	3
C_8	$MaximumBreakLength$	3
C_9	$OptimumBreakLength$	3
C_{10}	$MinimumLengthAfterDistance$	100
C_{11}	$NoExcess$	20
C_{12}	$NoShortage$	20
C_{13}	$FixedBreak$	10
C_{14}	$BreakQuantityInMeeting$	60

over ten runs for each instance we concluded that the random-walk strategy does indeed improve the results obtained. The solutions for 1% and 2.5% are consistently better than the ones computed with other probabilities.

4.3 Impact of Different Tabu Lists

In the next experiment we tried to find out which kind of tabu list returns the best solutions for our test instances. Again, we ran our tabu-search algorithm on ten instances, using the tabu lists storing inverse moves and breaks. For each kind of tabu list we considered different lengths, namely $2n$, $4n$, $6n$ and $8n$ for the tabu list type based on inverse moves and $\frac{1}{2}n$, n and $\frac{3}{2}n$ for the list type based on breaks, where 'n' always stands for the number of shifts. For both tabu list approaches we also considered variable lengths. The length of the variable tabu list with inverse moves ranges between $\frac{1}{2}n$ and $8n$, whereas the length of the variable list based on breaks ranges between $\frac{1}{2}n$ and $2n$. The lengths are adapted during the search as described in Section 3.5.

Regarding the average quality of ten runs, the most important point to notice is that there are only small differences between the average results obtained with different tabu lists. We may observe that longer tabu lengths, $8n$ for inverse moves and $\frac{3}{2}n$ for breaks, lead to slightly superior results. Furthermore we notice that in general using the tabu list prohibiting inverse moves returns solutions of slightly better quality.

4.4 Comparison of Different Local Search Strategies

In our final experiments we wanted to draw a comparison between different local search techniques to find out which method is best suited for the break scheduling problem.

Due to the best results of the previous experiments we chose algorithm min-conflicts-random-walk with a random walk probability of 2.5% (MCRW) and the tabu search with inverse moves and tabu list length $8n$. In addition we evaluated yet another combination of both variants, min-conflicts-random-walk with a random walk probability of 2.5% using a tabu list of length $8n$ (MCRW + TL). Table 5 reports for each instance and algorithm the average quality of the returned solutions and the

corresponding standard deviation. The reported averages are rounded to two decimal places.

Based on these results we come to the conclusion that min-conflicts-random-walk (column MCRW) and its combination with a tabu list (column MCRW + TL) outperform tabu search. Comparing MCRW with MCRW+TL we conclude that there is no significant difference in their performance.

Table 5 Average objective values and standard deviations returned by different local search methods for real-life instances.

	MCRW		Tabu Search		MCRW + TL	
	Average	SD	Average	SD	Average	SD
Basic_01	11594.04	0.00	11594.05	0.00	11594.04	0.00
Basic_02	11954.04	0.00	11954.04	0.00	11954.04	0.00
Basic_03	11954.04	0.00	11954.04	0.00	11954.04	0.00
Basic_04	5674.01	0.00	5746.61	33.00	5675.21	1.92
Basic_05	18834.21	0.00	18834.22	0.00	18834.21	0.00
Basic_06	10362.02	0.00	10363.22	2.16	10362.02	0.00
Basic_07	23182.23	0.00	23183.44	2.16	23182.23	0.00
Basic_08	3349.40	30.08	3612.01	64.00	3310.60	23.68
Basic_09	3971.41	43.80	4181.61	72.72	3988.41	39.68
Basic_10	2256.65	1.08	2261.75	2.28	2257.25	1.92
Basic_11	2434.03	0.00	2492.33	21.52	2434.03	0.00
Basic_12	1916.01	0.00	1917.22	1.92	1916.01	0.00
Fixed_01	2729.22	10.56	2739.02	32.80	2729.42	14.32
Fixed_02	2793.06	0.00	2817.07	24.00	2793.06	0.00
Fixed_03	2793.05	0.00	2793.06	0.00	2793.05	0.00
Fixed_04	2793.05	0.00	2793.06	0.00	2793.05	0.00
Fixed_05	2223.04	0.00	2223.05	0.00	2223.04	0.00
Fixed_06	3038.04	0.00	3062.04	58.40	3028.04	10.00
Fixed_07	2348.02	0.00	2383.02	37.00	2348.02	0.00
Meeting_01	3671.62	30.72	3770.22	53.84	3689.02	39.40
Meeting_02	9494.02	0.00	9494.03	0.00	9494.02	0.00
Meeting_03	19060.20	0.00	19204.22	114.40	19060.20	0.00
Meeting_04	23327.63	63.04	23639.54	139.44	23308.63	42.72

4.5 A Note on the Quality of the Obtained Solutions

Finally we want to illustrate that the solutions returned by our local search techniques are of acceptable quality, satisfying most of the constraints imposed on breaks and staffing. For that purpose we consider the best solution for instance Basic_12 obtained during our experiments.

In Figure 3 we see the curve of required employees (solid curve) in Basic_12 and the curve of working employees (dashed curve) resulting from the best solution found for that problem. The required minimum number of employees is not violated at any time. Table 6 presents the violation degree of each constraint for the best solution for instance Basic_12. We observe that nearly all constraints are completely satisfied. Only the optimum break length has been violated for some breaks and there exists some excess of working employees, which cannot be avoided due to the characteristics of

instance Basic_12. Moreover, for that particular instance the break lengths may not be further improved. Consequently, the quality of the computed solution is almost optimal.

Fig. 3 Curve of required and working employees resulting from the best solution for instance Basic_12.

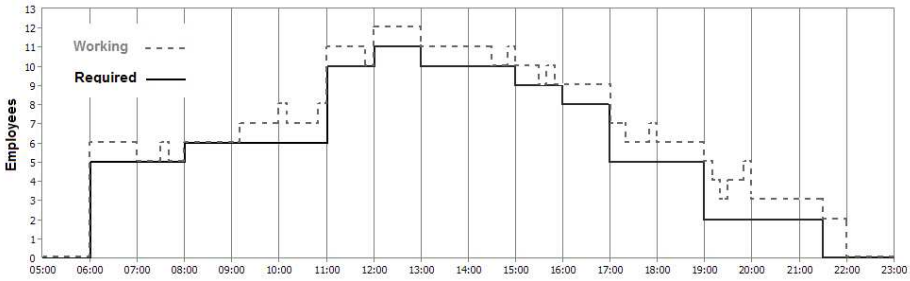


Table 6 Detailed results for instance Basic_12.

Constraint		Viol. Deg.	Weight	Product
C_1	MinimumDistanceToShiftBegin	0	10	0
C_2	MinimumDistanceToShiftEnd	0	10	0
C_3	MaximumDistanceToShiftBegin	0	100	0
C_4	MaximumDistanceToShiftEnd	0	100	0
C_5	MinimumDistanceBetweenBreaks	0	10	0
C_6	MaximumDistanceBetweenBreaks	0	100	0
C_7	MinimumBreakLength	0	3	0
C_8	MaximumBreakLength	0	3	0
C_9	OptimumBreakLength	12	3	36
C_{10}	MinimumLengthAfterDistance	0	100	0
C_{11}	NoExcess	94	20	1880
C_{12}	NoShortage	0	20	0
C_{SD}	NoSquaredDeviation	128	1/9576	0.01
	Total Violations		**1916.01**	

5 Previous Work

To the best of our knowledge the break scheduling problem addressed in this paper has not been presented in the literature yet. However several other similar problems concerning the design of shift plans with breaks have been addressed by different authors.

Break scheduling has been mainly considered in the literature as part of the shift scheduling problem. Dantzig developed the original set-covering formulation (Dantzig 1954). In this formulation there exists a variable for each feasible shift. Feasible shifts

are enumerated based on possible shift starts, shift lengths, breaks, and time windows for breaks. When the number of shifts increases rapidly this formulation is not practical. Bechtold and Jacobs proposed a new integer programming formulation (Bechtold and Jacobs 1990). In their formulation, the modeling of break placements is done implicitly. Authors reported superior results with their model compared to the set covering model. Their approach however is limited to scheduling problems of less than 24 hours per day. Thompson introduced a fully implicit formulation of shift scheduling problem (Thompson 1995). A comparison of different modeling approaches is given by Aykin (Aykin 2000).

A greedy algorithm for scheduling breaks after generating shifts has been presented in (Gärtner et al. 2004). The authors propose a simple algorithm which includes the phase of assigning the breaks greedily based on the information for the largest excess, and then applying simple repair steps on the assigned breaks.

Tellier and White present a tabu search based approach in order to solve a scheduling problem originating from call centers (Tellier and White 2006). They aim at minimizing the squared deviation of working employees from staffing requirements while various constraints are required to be satisfied. In (Tellier and White 2006) there is a correspondence between shifts and real employees of the contact center, and the constraints on a feasible solution restrict the position of breaks within shifts, the position and lengths of single shifts within the entire schedule, and the minimum and maximum number of paid working hours per employee. Canon investigates also the use of tabu search for the shift design problem including breaks (Canon 2007).

Unfortunately a direct comparison between our approach and the approaches we just mentioned cannot be done due to the differences in problem formulations and constraints.

6 Conclusions

In this paper we investigated the break scheduling problem, a complex, real-life problem of high practical relevance. We proposed several different heuristics to solve this problem: a tabu search with two different kinds of tabu lists as well as a min-conflicts based heuristic. Moreover we considered the hybridization of the min-conflicts heuristic with a random-walk strategy and tabu lists.

After designing and implementing these methods we used them to find solutions for 23 real-life problem instances. The obtained results lead us to the following conclusions:

- ▷ Our min-conflicts approach is improved by the random-walk strategy.
- ▷ Min-conflicts search in combination with random walk is both, better suited to this problem than our tabu search implementation.
- ▷ In general, the proposed methods returned high quality solutions for the regarded real-life examples.

For future work we will extend our methods to solve similar problems appearing in other working areas than call centers. Furthermore we will investigate the application of other meta-heuristic methods to solve break scheduling problems.

Acknowledgements The research herein 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).

We furthermore want to thank Pascal van Hentenryck and Laurent Michel for answering our questions concerning COMET.

References

Aykin, T. (2000). A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125:381 –397.

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

Canon, C. (2007). Personnel scheduling in the call center industry. *4OR: A Quarterly Journal of Operations Research*, 5(1):89–92.

Dantzig, G. (1954). A comment on eddie's traffic delays at toll booths. *Operations Research*, 2:339 –341.

Gärtner, J., Musliu, N., and Slany, W. (2004). A heuristic based system for generation of shifts with breaks. In *Proceedings of the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (Springer) Cambridge*.

Gaspero, L. D., Gärtner, J., Musliu, G. K. N., Schaerf, A., and Slany, W. (2007). The minimum shift design problem. *Annals of Operations Research*, 155:79–105.

Glover, F. and Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers.

Hentenryck, P. V. and Michel, L. (2005). *Constraint-Based Local Search*. The MIT Press, Massachusetts.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205.

Musliu, N., Schaerf, A., and Slany, W. (2004). Local search for shift design. *European Journal of Operational Research*, 153(1):51–64.

Selman, B., Kautz, H. A., and Cohen, B. (1993). Local search strategies for satisfiability testing. In *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*.

Tellier, P. and White, G. (2006). Generating personnel schedules in an industrial setting using a tabu search algorithm. *E. K. Burke, H. Rudov (Eds.): PATAT 2006*, pages 293–302.

Thompson, G. (1995). Improved implicit modeling of the labor shift scheduling problem. *Management Science*, 41(4):595–607.