

# The break scheduling problem: complexity results and practical algorithms

Magdalena Widl · Nysret Musliu

Received: 27 May 2013 / Accepted: 3 March 2014 / Published online: 25 March 2014  
© Springer-Verlag Berlin Heidelberg 2014

**Abstract** Break scheduling problems arise in working areas where breaks are indispensable, e.g., in air traffic control, supervision, or assembly lines. We regard such a problem from the area of supervision personnel. The objective is to find a break assignment for an existing shiftplan such that various constraints reflecting legal demands or ergonomic criteria are satisfied and such that staffing requirement violations are minimised. We prove the NP-completeness of this problem when all possible break patterns for each shift are given explicitly as part of the input. To solve our problem we propose two variations of a memetic algorithm. We define genetic operators, a local search based on three neighbourhoods, and a penalty system that helps to avoid local optima. Parameters influencing the algorithms are experimentally evaluated and assessed with statistical methods. We compare our algorithms, each with the best parameter setting according to the evaluation, with the state-of-the-art algorithm on a set of 30 real-life and randomly generated instances that are publicly available. One of our algorithms returns improved results on 28 out of the 30 benchmark instances. To the best of our knowledge, our improved results for the real-life instances constitute new upper bounds for this problem

**Keywords** Break scheduling · Complexity · Memetic algorithms · Hybrid genetic algorithms · Real-life application

---

M. Widl (✉)  
Knowledge-based Systems Group, Vienna University of Technology, Favoritenstrasse 9, 1040 Vienna, Austria  
e-mail: widl@kr.tuwien.ac.at

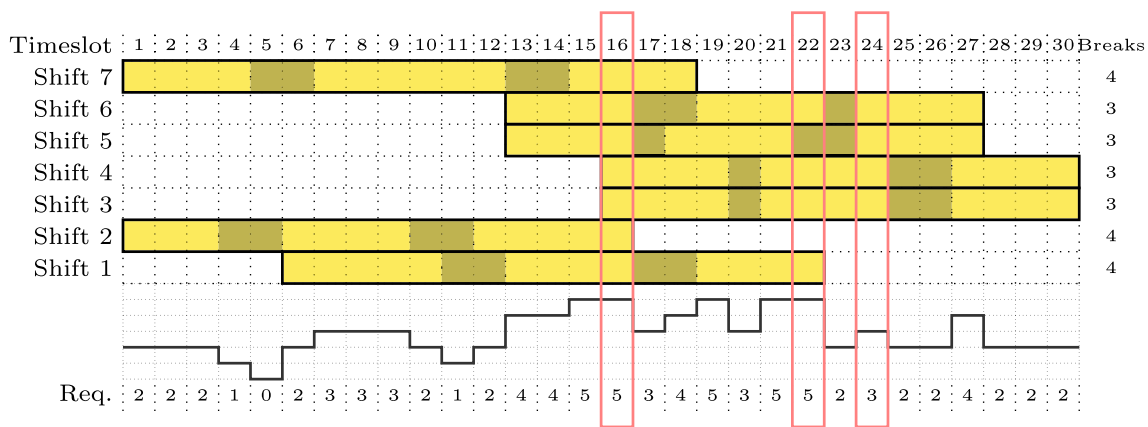
N. Musliu  
Databases and Artificial Intelligence Group, Vienna University of Technology, Favoritenstrasse 9, 1040 Vienna, Austria  
e-mail: musliu@dbai.tuwien.ac.at

## 1 Introduction

Many working areas require staff members to maintain high concentration while performing their tasks. These include air traffic control, security checking, supervision, or assembly line workers, where loss of concentration can result in dangerous situations. It is therefore required that staff take breaks after given periods of time. Additionally, staffing requirements, which define the number of staff required to be working during a given period, should be fulfilled.

Our particular problem originates from a real-life scenario in the area of supervision personnel. As input we are given a *shiftplan* consisting of consecutive timeslots and of scheduled shifts, the total *breaktime* required for each shift, a set of *temporal constraints* concerning the locations and lengths of breaks and of working periods, and *staffing requirements* for each timeslot. The breaktime for each shift is to be scheduled such that the temporal constraints are satisfied and violations of staffing requirements are minimised. We denote our formulation as break scheduling problem (BSP). Figure 1 depicts a small shiftplan with a possible solution.

Previously, the task of scheduling of breaks has been addressed mainly as part of the so-called shift scheduling problem. Several approaches have been proposed for problem formulations that include a small number of breaks. These approaches schedule the breaks within a shift scheduling process. In particular, Dantzig developed the original set-covering formulation [11] for the *shift scheduling problem*, in which feasible shifts are enumerated based on possible shift starts, shift durations, breaks, and time windows for breaks. Examples of integer programming formulations for shift scheduling include [1, 3, 29]. A comparison of different modeling approaches was given by Aykin [2]. Rekik et al. [25] developed two other implicit models and improved upon previous approaches, among them Aykin's original



**Fig. 1** A shiftplan with different shifts and breaktimes. Breaks are scheduled such that long shifts contain four breaks, short shifts contain three breaks, no break is longer than two timeslots, and no working period is longer than six timeslots. The staffing requirements are indi-

cated as a demand curve and as numbers of staff in the bottom line. They are not satisfied in all timeslots, i.e., there is undercover in timeslot 22 and overcover in timeslots 16 and 24

model. Tellier and White [28] developed a tabu search algorithm to solve a shift scheduling problem originating in contact centers. This algorithm has been integrated into the workforce scheduling system Contact Center Scheduling 5.5. An approach [15] to shift scheduling with breaks suggests to first schedule the shiftplan without breaks (see [13,21]) and then generate three to four breaks per shift with a greedy approach. The formulation of a shift scheduling problem with a planning period of 1 day and at most three breaks (two 15 min breaks and a lunch break of 1 h) has been considered recently in [8,24]. In [8], the authors make use of automata and context-free grammars to formulate constraints on sequences of decision variables. The approach suggested in [24] is based on modeling the regulations of the shift scheduling problem by using regular and context-free languages. Then a large neighborhood search is applied to find solutions for the whole problem of scheduling both shifts and breaks. In addition to the previous model, the authors apply their methods for single and multiple activity shift scheduling problems. A new implicit formulation for multi-activity shift scheduling problems using context-free grammars has been proposed by Côté et al. [9].

Some important break scheduling problems arising in call centers, airports, and other areas include a much higher number of breaks compared to the problem formulations in previous works on shift scheduling. Also, additional requirements like time windows for lunchbreaks or restrictions on the length of breaks and worktime can emerge. These new constraints significantly enlarge the search space. Therefore, researchers recently started to consider a new approach which regards shift scheduling and break scheduling as two different problems and tries to solve them in separate phases.

The problem tackled in this paper (BSP) consists of scheduling breaks to a given shiftplan. It has first been intro-

duced by Beer et al. [6]. In addition to an arbitrary number of breaks per shift, the problem formulation allows to impose several specific temporal constraints on the assignment of breaks. A similar break scheduling problem, which origins in call centers and also includes meetings and some slightly different constraints, has been described in [5,26]. Beer et al. [6] also introduced real-life benchmark instances containing shifts that include more than ten breaks. To solve BSP, local search techniques based on min-conflicts have been proposed in [4,6]. A simple memetic algorithm for this problem has been proposed in [22]. An algorithm based on constraint programming and local search for BSP in combination with shift design has been investigated in [16]. Although initial solutions of this hybrid solver are promising, the results obtained by solving the break scheduling problem separately (after generation of shifts) could not be improved.

These approaches to BSP have been used successfully for solving large real-life instances and have also been deployed in practice. However, the best solutions for these instances are not yet known, and the question is whether the solutions can be improved. Further, the computational complexity of this problem has not been investigated so far.

This paper presents new complexity results for this problem. In particular, we prove that the decision variant of BSP is NP-complete when all possible break patterns for each shift are defined explicitly as part of the input. To obtain improved upper bounds for the BSP, we propose two new memetic algorithms. Both algorithms are based on the same initialisation process and the same memetic representation, and apply a local search based on the same set of three neighbourhoods. The local search contributes through hill-climbing rather than diversification. The first algorithm is based on a classic memetic approach with crossover, mutation, and selection. The second algorithm contains new ideas

that avoid the weaknesses of the first approach. These new ideas include a penalty system and a new crossover operator.

Both algorithms depend on a set of parameters for which we experimentally evaluate different values. The impact of each parameter is statistically verified. We finally compare the outcomes of our algorithms with the best existing results for a set of benchmarks from the literature. One of our algorithms returns improved results on 28 out of 30 instances. To the best of our knowledge, these results represent new upper bounds for the available BSP real-life instances.

The problem definition, the second algorithm, and the evaluation of this algorithm have been presented at the 7th International Workshop on Hybrid Metaheuristics [31]. This paper extends [31] by a complexity analysis that justifies the application of metaheuristics, a detailed explanation of a fast method to generate break patterns, an additional algorithm with a parameter evaluation and a discussion of its weaknesses, and a comparison of the two algorithms to methods presented in the literature.

The remaining parts of this work are organised as follows: we first give a formal definition of BSP and present our complexity results. Solving BSP with memetic algorithms is presented in Sects. 4, 5 and 6, where Sect. 4 describes elements the two algorithms have in common, and Sects. 5 and 6 describe each algorithm with its parameter evaluation. Section 7 describes the set of real-life and random instances, the experimental setup, and a comparison with the literature. We draw our conclusions and describe the future work in Sect. 8.

## 2 Problem statement

BSP deals with scheduling of breaks in a *shiftplan* that consists of consecutive *timeslots* and of *shifts* starting and ending in defined timeslots. One shift represents exactly one employee on duty within a sequence of timeslots. Two or more shifts may overlap in time, i.e., have timeslots in common. A timeslot in a particular shift is referred to as *slot*. To each slot, either a break, worktime, or time used for familiarisation with a new working situation has to be assigned. The latter stems from the real-life nature of BSP. After a break, the working situation may have changed, and therefore the employee is given some time to get familiar with the new circumstances (for example, in air traffic control). We are further given staffing requirements which indicate the number of employees required to be working in each timeslot.

The objective is to find an assignment for each slot such that breaks are distributed within each shift according to some temporal constraints and such that violations of staffing requirements are minimised. These violations can occur as over- or undercover violation. Since we are dealing with a real-life problem, different measures are taken for the two

types, as in the particular domain, undercover is a more serious problem than overcover.

In the following, we provide a set of formal definitions necessary to provide a precise problem statement.

**Definition 1** (*Shiftplan*  $\mathcal{P}$ ) A pair  $(T, S)$  where  $T = \{1, 2, \dots, k\}$  is a set of consecutive timeslots and  $S = \{S_1, \dots, S_n\}$  is a set of shifts.

**Definition 2** (*Timeslot*  $t$ ) An element of  $T$  representing a time period of fixed length. In our real-life instances, each timeslot corresponds to a period of 5 min.

**Definition 3** (*Shift*  $S$ ) A set  $S = \{t_i, t_{i+1}, \dots, t_{i+m}\}$ ,  $S \subseteq T$ , of consecutive timeslots, i.e.,  $t_{j+1} - t_j = 1$  for  $i \leq j < i+m$ . The shift start is denoted  $S_s = t_i$  and the shift end  $S_e = t_{i+m}$ . Each shift represents exactly one employee on duty. Two or more shifts can have timeslots in common.

**Definition 4** (*Slot*) A timeslot in a particular shift. A slot can be assigned one of three values: 1 (*I-slot*) for a working employee, 0 (*O-slot*) for an employee on break or  $\bar{0}$  ( $\bar{O}$ -slot).  $\bar{0}$ -slots are assigned to those and only those slots that directly follow a sequence of 0-slots. A  $\bar{0}$ -slot stands for an employee who is getting familiar with an altered working situation after a break. During a  $\bar{0}$ -slot, the employee is not consuming breaktime but neither counted as working staff regarding staffing requirements.

**Definition 5** (*Breaktime*  $\tau(|S|)$ ) A function  $\tau : \{|S_1|, \dots, |S_n|\} \rightarrow \mathbb{N}$  that maps each shift length to a number of 0-slots that have to be assigned to a shift  $S_i$  with length  $|S_i|$ .

**Definition 6** (*Staffing requirements*  $\rho(t)$ ) Function  $\rho : T \rightarrow \mathbb{N}$  assigning a number of required 1-slots to each timeslot.

**Definition 7** (*Work period*  $W$ ) A set of consecutive 1- and  $\bar{0}$ -slots in a particular shift.

**Definition 8** (*Break*  $B$ ) A set of consecutive 0-slots in a particular shift.

**Definition 9** (*Temporal constraints*  $\mathcal{C}$ ) A set  $\mathcal{C} = \{C_1, \dots, C_5\}$  of global restrictions regarding lengths and locations of breaks and work periods inside shifts.

$C_1$  Break positions  $(d_1, d_2)$ . In each shift, each timeslot in  $\{t_i, t_{i+1}, \dots, t_{i+d_1-1}\}$  and  $\{t_{i+m-d_2+1}, \dots, t_{i+m}\}$  must be assigned a 1-slot, i.e., a break may start earliest  $d_1$  timeslots after the start and end latest  $d_2$  timeslots before the end of its associated shift.

$C_2$  Lunch breaks  $(h, g, l_1, l_2)$ . Each shift  $S$  with  $|S| > h$  must contain a break  $B_L$  with  $B_L \geq g$ , i.e., with a minimum length of  $g$  timeslots, starting at least  $l_1$  and ending at most  $l_2$  timeslots after the start of its shift.

- C<sub>3</sub> Work periods  $(w_1, w_2)$ . For each work period  $W$ ,  $w_1 \leq |W| \leq w_2$ .
- C<sub>4</sub> Minimum break duration  $(w, b)$ . A work period  $W$  with  $|W| \geq w$  must be followed by a break  $B$  with  $|B| \geq b$ .
- C<sub>5</sub> Break lengths  $(b_1, b_2)$ . For each break  $B$ ,  $b_1 \leq |B| \leq b_2$ .

**Definition 10** (Break pattern  $D$ ) A set  $D \subset S$  of timeslots representing a set of breaks (0-slots) for a shift  $S$  such that  $|D| = \tau(|S|)$  and all constraints in  $\mathcal{C}$  are satisfied.

**Definition 11** (Possible break patterns  $\mathcal{D}_S$  for a shift  $S$ ) A set  $\mathcal{D}_S \subset 2^S$  of break patterns for shift  $S$ . Sect. 4.2 explains how a set of break patterns can be generated.

**Definition 12** (Solution  $\mathcal{B}$ ) A total map  $\mathcal{B} : \mathcal{S} \rightarrow 2^T$  with  $\mathcal{B}(S) \in \mathcal{D}_S$  for each  $S \in \mathcal{S}$ . A solution assigns a break pattern to each shift.

Based on these definitions we define BSP as follows:

**Definition 13** (Break scheduling problem BSP) Instance A tuple  $(\mathcal{P}, \tau, \rho, \mathcal{C})$  with each element as described above. Objective Let  $\mathcal{Q} = (\mathcal{P}, \tau, \rho, \mathcal{C})$  be an instance of BSP. The objective is to find a solution  $\mathcal{B}$  such that the following objective function is minimised:

$$F(\mathcal{B}, T, \rho) = w_o \cdot O(\mathcal{B}, T, \rho) + w_u \cdot U(\mathcal{B}, T, \rho)$$

where

- $w_o$  and  $w_u$  are weights for over- and undercover violations respectively, and
- for  $\omega(\mathcal{B}, t)$  the number of 1-slots in timeslot  $t \in T$  according to  $\mathcal{B}$ ,
- $U(\mathcal{B}, T, \rho) = \sum_{t \in T} \max(0, \rho(t) - \omega(\mathcal{B}, t))$ , i.e., the undercover violations, and
- $O(\mathcal{B}, T, \rho) = \sum_{t \in T} \max(0, \omega(\mathcal{B}, t) - \rho(t))$ , i.e., the overcover violations.

Figure 2 depicts a solution for a small instance of BSP.

### 3 Computational complexity

We present a proof of NP-completeness for BSP under the condition that break patterns are given explicitly as part of the input.

BSP can be re-formulated as decision problem with the same input  $(\mathcal{P}, \tau, \rho, \mathcal{C})$ . The question is whether for an instance  $\mathcal{Q}$  there exists a solution  $\mathcal{B}$  such that  $F(\mathcal{B}, T, \rho) = 0$ .

**Lemma 1** The problem BSP is in NP.

*Proof* Given an instance  $\mathcal{Q} = (\mathcal{P}, \tau, \rho, \mathcal{C})$  of BSP and a set  $\mathcal{B}$  containing an arbitrary break pattern for each  $S \in \mathcal{S}$ , it can

be checked in time  $O(|T| \cdot |\mathcal{S}|)$  whether  $\rho(t)$  is satisfied for each timeslot  $t$ . □

We define BSP' as modification of BSP as follows:

**Definition 14** (BSP') Without loss of generality, we eliminate the 0-slots from BSP so that break patterns in BSP' contain only 0- and 1-slots. An instance of BSP is a tuple  $\mathcal{Q}' = (\mathcal{P}, \tau, \rho, \gamma)$  where the definitions of  $\mathcal{P}, \tau, \rho$  equal those in BSP and  $\gamma$  is a function that maps each shift  $S$  to a set  $\mathcal{D}'_S \subset 2^S$  of break patterns such that for each  $D' \in \mathcal{D}'_S$  it holds that  $|D'| = \tau(S)$ . The question is whether for an instance  $\mathcal{Q}'$  there exists a solution  $\mathcal{B}'$  such that  $F(\mathcal{B}', T, \rho) = 0$ .

The difference between BSP and BSP' is that BSP is given the set of possible break patterns implicitly by  $\tau(S)$  and  $\mathcal{C}$  whereas for BSP' this set is given explicitly by  $\gamma$ .

We show that BSP' is NP-complete by reduction from the well-known NP-complete problem Exact Cover by 3-Sets (X3C) [14].

**Definition 15** (X3C) An instance of X3C is a pair  $\mathcal{X} = (U, \mathcal{F})$  where  $U$  is a set with  $|U| = 3m, m > 1$ , and  $\mathcal{F}$  is a collection of three-element subsets of  $U$ , i.e.,  $|F| = 3$  and  $F \subset U$  for each  $F \in \mathcal{F}$ . The question is whether  $\mathcal{F}$  contains an exact cover for  $U$ , i.e., a subcollection  $\mathcal{F}' \subseteq \mathcal{F}$ , such that each element of  $U$  occurs in exactly one member of  $\mathcal{F}'$ .

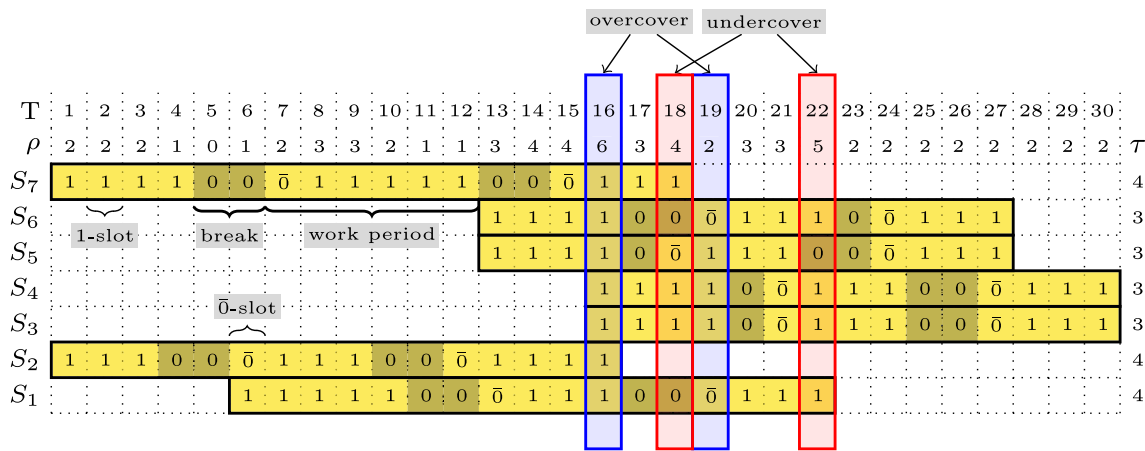
**Theorem 1** The problem BSP' is NP-complete.

*Proof* The NP-membership of BSP' follows from the NP-membership of BSP. To show that BSP' is NP-hard, we present a reduction from X3C. Given an arbitrary instance  $\mathcal{X} = (U, \mathcal{F})$  of X3C, we construct an instance  $\mathcal{Q}' = (\mathcal{P}, \tau, \rho, \gamma)$  of BSP' in polynomial time as follows.

- $\mathcal{P} = (T, \mathcal{S})$  where  $T = \{1, 2, \dots, 3m\}, \mathcal{S} = \{S_1, S_2, \dots, S_m\}$ , and  $S = T$  for each  $S \in \mathcal{S}$ .
- $\tau(|S|) = 3$  for each  $S \in \mathcal{S}$ .
- $\rho(t) = |S| - 1$  for each  $t \in T$ , i.e., in each timeslot exactly one break is required.
- $\gamma(S) = \{\{\sigma(u) \mid u \in F\} \mid F \in \mathcal{F}\}$  where  $\sigma : U \rightarrow \mathbb{N}$  is a bijective function enumerating all elements in  $U$ :  $\sigma(u_1) = 1, \sigma(u_2) = 2, \dots, \sigma(u_{3m}) = 3m$ .

The following observations can be made from the construction of the instance of  $\mathcal{Q}'$ . The number of timeslots equals the number of elements in  $U$ . All shifts are of the same length and contain all timeslots. The number of 0-slots for each shift is 3, which is the size of each  $F \in \mathcal{F}$ . The number of shifts is  $m$ , which in  $\mathcal{X}$  is the number of sets needed to cover  $U$ .

For each timeslot  $\rho$  requires one 0-slot. All shifts  $S \in \mathcal{S}$  share the same set  $\gamma(S)$  of break patterns, which equals the elements of  $\mathcal{F}$  mapped by  $\sigma$ .



**Fig. 2** An instance of BSP with a solution. A solution for instance:  $(P, \tau, \rho, C)$  with  $P = (S, T)$ ,  $T = \{1, \dots, 30\}$ ,  $S = \{S_1, S_2, \dots, S_7\}$ ,  $\tau(|S|) = 3$  if  $|S| \leq 15$ ;  $\tau(|S|) = 4$  otherwise,  $\rho$  as stated in the sec-

ond line,  $C_1 = (3, 3)$ ,  $C_2 = (25, 4, 7, 7)$ ,  $C_3 = (3, 6)$ ,  $C_4 = (5, 2)$ ,  $C_5 = (1, 3)$ . The solution depicted for this instance is the mapping  $B(S_1) = \{11, 12, 17, 18\}$ ,  $B(S_2) = \{4, 5, 10, 11\}$ , etc

We now show that  $\mathcal{X}$  is a positive instance of X3C if and only if  $\mathcal{Q}'$  is a positive instance of  $BSP'$ .

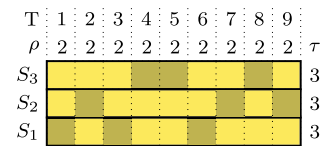
$\Rightarrow$  If  $\mathcal{X}$  is a positive instance of X3C, then there exists a collection of sets  $\mathcal{F}'$  such that each element  $u \in U$  occurs in one  $F \in \mathcal{F}'$ . By applying  $\sigma$  to all elements in each set  $F \in \mathcal{F}'$  we obtain a set of sets that represent break patterns for the shifts in the instance  $\mathcal{Q}'$ . These break patterns cover exactly the timeslots in  $T$  because  $\mathcal{F}'$  covers exactly the elements in  $U$  and because of the definition of  $T$ . Further, the staffing requirement function  $\rho$  is defined to return  $3m - 1$  for any timeslot, therefore, by exactly covering  $T$ , the break patterns also fulfill the staffing requirements. Therefore,  $\mathcal{Q}'$  is a positive instance of the reduced  $BSP'$  problem.

$\Leftarrow$  If  $\mathcal{Q}'$  is a positive instance of  $BSP'$ , then there exists a set of break patterns such that each timeslot in  $T$  occurs in exactly one break pattern because by the definition of  $\rho$ , each timeslot must contain exactly one 0-slot. By applying  $\sigma^{-1}$  to each element of each break pattern, we thus obtain a set of sets that cover exactly the elements in  $U$ . Therefore,  $\mathcal{X}$  is a positive instance of the original X3C problem.  $\square$

In the following, we present an example for the reduction from an X3C problem to  $BSP'$ .

Let  $\mathcal{X} = (U, \mathcal{F})$  with  $U = \{A, B, C, D, E, F, G, H, I\}$  and  $\mathcal{F} = (\{A, C, F\}, \{C, D, E\}, \{F, G, H\}, \{A, D, G\}, \{B, G, I\}, \{D, E, H\})$ . Then an instance  $\mathcal{Q}' = (P, \tau, \rho, \gamma)$  of  $BSP'$  is constructed as follows:

- $T = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\mathcal{S} = \{S_1, S_2, S_3\}$  with  $S_1 = S_2 = S_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\rho(t) \begin{array}{c|c|c|c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{array}$



**Fig. 3** Solution for  $BSP'$  instance reduced from X3C instance

- $\sigma \begin{array}{c|c|c|c|c|c|c|c|c} U & A & B & C & D & E & F & G & H & I \\ \hline T & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$
- $\gamma(S_1) = \gamma(S_2) = \gamma(S_3) = (\{1, 3, 6\}, \{3, 4, 5\}, \{6, 7, 8\}, \{1, 4, 7\}, \{2, 7, 9\}, \{4, 5, 8\})$

A solution for  $\mathcal{Q}'$  is the mapping  $S_1 \mapsto \{1, 3, 6\}$ ,  $S_2 \mapsto \{2, 7, 9\}$ ,  $S_3 \mapsto \{4, 5, 8\}$ . Figure 3 depicts this solution. It is easy to construct the solution for the X3C instance  $\mathcal{X}$  by looking up the values in the  $\sigma$  function.

### 4 New memetic algorithms for BSP

Motivated by the complexity results of the previous section, we propose two different memetic algorithms to solve the BSP. Memetic algorithms were first described in [20]. An overview on memetic algorithms for scheduling problems, among others, is given in [10]. An example of a memetic approach to nurse scheduling is given by [7].

A memetic algorithm consists of genetic operators and local improvements executed on a set of solutions. The initial set of solutions is usually created randomly or by a fast heuristic. Our algorithms both use the same memetic representation, initialisation heuristic, and neighbourhoods for a local search. They differ in their genetic operators, the application

of the local search, and one of them additionally applies a penalty system.

We earlier proposed a simple memetic algorithm for this problem [22]. The two algorithms in this work are a significant improvement regarding several aspects. In particular, a new memetic representation based on time periods rather than on single shifts contributes to the improvements. This new representation also requires different genetic operators. The previous algorithm is clearly outperformed by the current algorithms on all benchmark instances from the literature and therefore it is not described here. The algorithm presented in Sect. 6 of this work has appeared previously in conference proceedings of the International Workshop on Hybrid Metaheuristics [31].

In this section we present the components the two algorithms have in common, i.e., the memetic representation, the initialisation of sets of break patterns and of an initial population, the neighbourhoods, and the local search procedure. Sections 5 and 6 describe the two algorithms, their parameters, and the evaluation of the parameters.

### 4.1 Representation

**Definition 16 (Memetic representation)** The memetic representation of an instance of BSP is a set  $\bar{\mathcal{M}} = \{\bar{M}_1, \dots, \bar{M}_q\}$  of memes. The memetic representation of a solution of an instance of BSP is a set  $\mathcal{M} = \{M_1, \dots, M_q\}$  of instances of memes, or alleles in genetic terms.

**Definition 17 (Meme  $\bar{M}$  and meme instance  $M$ )** A meme  $\bar{M} \in \bar{\mathcal{M}}$  is a triple  $(t_s, t_e, \mathcal{S}_M)$  where

- $t_s, t_e \in T, t_s < t_e$
- $\mathcal{S}_M \subseteq \mathcal{S}$ , and

- $\mathcal{S}_M$  contains a shift  $S$  if and only if  $t_s \leq \lfloor (S_s + S_e)/2 \rfloor < t_e$  with  $S_s$  and  $S_e$  denoting the first, respectively last, timeslot of the shift.

An instance  $M$  of a meme  $\bar{M}$  carries parts of a solution with respect to the time period between  $t_s$  and  $t_e$ . It is represented by a tuple  $(t_s, t_e, \mathcal{S}_M, \mathcal{B}_M, F_M)$  where

- $t_s, t_e$ , and  $\mathcal{S}_M$  are defined as in  $\bar{M}$ ,
- $\mathcal{B}_M$  is a function  $\mathcal{B}_M : \mathcal{S}_M \rightarrow 2^T$  with  $\mathcal{B}(S) \in \mathcal{D}_S$  for each  $S \in \mathcal{S}_M$ . It assigns a break pattern to each shift involved in  $M$ , and
- $F_M = F(\mathcal{B}_M, T_M, \rho)$  is a fitness value with  $F$  as in Definition 13 and  $T_M = \bigcup_{S \in \mathcal{S}_M} S$  contains the timeslots involved in  $M$ .

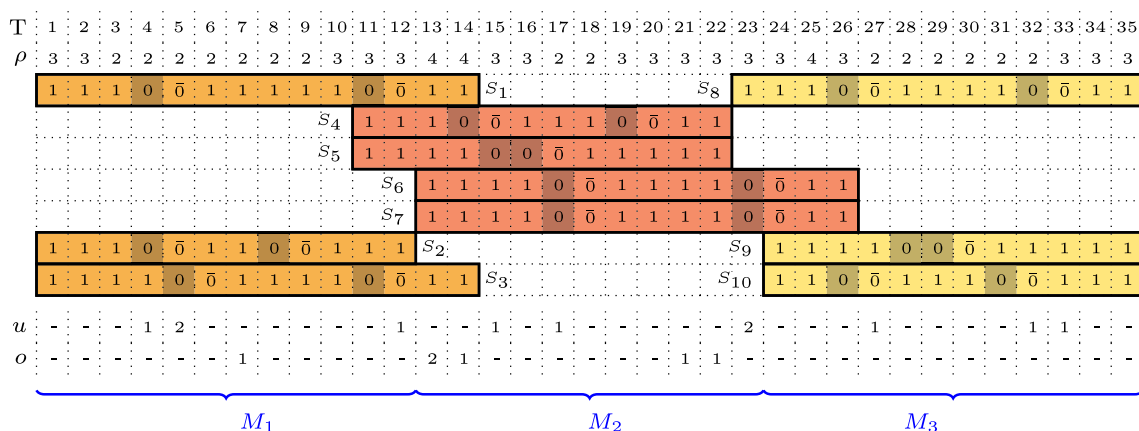
Figure 4 depicts the memetic representation of a solution for an instance of BSP.

We retrieve the set  $\bar{\mathcal{M}}$  of memes for an instance  $Q = (\mathcal{P}, \tau, \rho, \mathcal{C})$  of BSP heuristically as follows.

For each  $t \in T$  let set  $\mathcal{S}_t = \{S \in \mathcal{S} \mid t \in S\}$ , i.e., the set of shifts taking place during  $t$ . Further let  $p : T \rightarrow \mathbb{N}$  be a function assigning a value to each timeslot such that

$$p(t) = \sum_{S \in \mathcal{S}_t} \begin{cases} 0 & \text{if } t < S_s + d_1 \\ 0 & \text{if } t > S_e - d_2 \\ 1 & \text{if } S_s + d_1 < t < S_s + d_1 + b_1 + w_1 \\ 1 & \text{if } S_e - d_2 > t > S_e - d_2 - b_1 - w_1 \\ 100 & \text{otherwise} \end{cases}$$

As described in Sect. 2,  $d_1$  and  $d_2$  denote the number of timeslots after  $S_s$  and before  $S_e$  respectively, to which no



**Fig. 4** Memetic representation. The memetic representation of a shift-plan with solution by a set of meme instances  $\mathcal{M} = \{M_1, M_2, M_3\}$  where  $M_1 = (1, 12, \{S_1, S_2, S_3\}, \langle S_1 \mapsto \{4, 11\}, S_2 \mapsto \{4, 8\}, S_3 \mapsto \{5, 11\} \rangle, 48)$ ,  $M_2 = (13, 23, \{S_4, S_5, S_6, S_7\}, \langle S_4 \mapsto \{14, 19\}, S_5 \mapsto \{15, 16\}, S_6 \mapsto \{17, 23\}, S_7 \mapsto \{17, 23\} \rangle, 60)$ , and  $M_3 =$

$(24, 35, \{S_8, S_9, S_{10}\}, \langle S_8 \mapsto \{26, 32\}, S_9 \mapsto \{28, 29\}, S_{10} \mapsto \{26, 31\} \rangle, 48)$ . The values for  $F_M$  are calculated according to Definitions 13 and 17 with weights  $w_o = 2$  and  $w_u = 10$ . Lines  $u$  and  $o$  show the number of undercover and overcover violations per timeslot respectively

breaks can be assigned,  $b_1$  stands for the minimal length of a break, and  $w_1$  for the minimal length of a work period.

The lower  $p(t)$  for a timeslot  $t$ , the less breaks can be assigned to  $t$ .  $p(t)$  is defined and the constants 0, 1, and 100 are chosen in a way such that timeslots with many break assignment possibilities are well distinguished. In an alternative setting, the constants could be replaced by variables depending on the number of possible break patterns of  $S_t$ .

The objective is to find a set  $\bar{T}$  of timeslots which have a low number of possible breaks and whose pairwise distance is above a certain threshold. These timeslots serve as “borders” between memes. In between these borders, we will find timeslots with a high value for  $p$ . Shifts sharing such timeslots will be assigned to the same meme.

To this end, we determine an ordered set of timeslots  $\bar{T} \subset T$  such that

- $\sum_{t \in \bar{T}} p(t)$  is minimised
- For each  $t_i, t_j \in \bar{T}$ :  $|t_i - t_j| > d$  with  $d = \lfloor \frac{\min_{S \in \mathcal{S}} |S|}{2} \rfloor$ , i.e., the distance between each pair of timeslots is at least half of the smallest shift length

To retrieve this set, we start by adding timeslot  $t_0$  to  $\bar{T}$  such that  $p(t_0)$  returns the smallest value among the elements in  $T$ , ties broken randomly. We continue by adding timeslots  $t_i$  for  $i > 0$  to  $\bar{T}$  such that  $t_i \in T \setminus \bar{T}$ ,  $p(t_i)$  is smaller than the value for any other  $t \in T \setminus \bar{T}$ , and  $|t_i - t_j| > d$  for all  $t_l, t_j \in \bar{T} \cup \{t_i\}$  until no more timeslots exist which fulfill the last requirement. For  $k = |\bar{T}| - 1$  we retrieve the set  $\bar{M} = \{\bar{M}_1, \bar{M}_2, \dots, \bar{M}_k\}$  of memes with  $\bar{M}_i = (t_i, t_{i+1}, S_M)$  for  $0 \leq i < k$  and  $S_M$  as described in Definition 17.

**Definition 18 (Individual  $I$ )** A pair  $I = (\mathcal{B}, F_I)$  where  $\mathcal{B}$  is a solution for an instance of BSP and  $F_I$  the fitness value of the solution as described in Definition 13.

**Definition 19 (Population  $\mathcal{I}$ )** A set  $\mathcal{I}$  of individuals.

**Definition 20 (Generation)** State of a population during an iteration of the algorithm.

**Definition 21 (Memepool  $\hat{\mathcal{M}}$ )** The set  $\hat{\mathcal{M}}$  of all meme instances in a generation.

**Definition 22 (Elitist  $E$ )** The individual  $E \in \mathcal{I}$  with the lowest fitness value among a generation.

#### 4.2 Initialisation of break patterns

Both presented algorithms rely on a set of break patterns that is computed at the beginning. The problem of finding a single break pattern  $D \in \mathcal{D}_S$  for a shift  $S$  can be modeled as a simple temporal problem [12] and consequently be solved in cubic

time with respect to  $|S|$  using Floyd–Warshall’s shortest path algorithm [23], but the number of break patterns  $|\mathcal{D}_S|$  usually grows exponentially with respect to  $|S|$ . However, with some restrictions on the break patterns and by exploiting some of their common characteristics, we can generate big sets of break patterns using a reasonable amount of computing time and space.

We precalculate a subset  $\hat{\mathcal{D}}_S$  of  $\mathcal{D}_S$  for each shift length with the following restrictions on the break patterns in  $\hat{\mathcal{D}}_S$ .

- For  $C_5$ , which denotes the minimal and maximal allowed length of breaks, we replace  $b_2$  by  $\hat{b}_2 = \min(b_2, b_1 + 1)$ , except for lunchbreaks, where we replace  $b_1$  and  $b_2$  by  $\hat{b}_1 = \hat{b}_2 = g$ , i.e., their minimal value according to  $C_2$ .
- Break patterns can include breaks of different sizes. For example, if  $\tau(|S|) = 10$  without lunchbreak, then this breaktime can be constructed out of two breaks with length 2 and two breaks with length 3, or five breaks with length 2. Thus, possible break patterns in  $\mathcal{D}_S$  for a shift  $S$  include all possible combinations of break lengths that sum up to  $\tau(|S|)$  as well as their permutations. In the example with  $\tau(|S|) = 10$ , the following are all possible permutations of break lengths: (3, 3, 2, 2), (2, 3, 2, 3), (3, 2, 3, 2), (2, 2, 3, 3), (2, 3, 3, 2), (3, 2, 2, 3) and (2, 2, 2, 2, 2). We select only one permutation from each combination at random. For  $\tau(|S|) = 10$ , we compute all possible patterns with break combination (2, 2, 2, 2, 2) and all possible patterns for one permutation selected randomly from all permutations of the combination (3, 3, 2, 2).

To save computation time and space, we exploit the fact that shifts and periods within shifts of the same length share the same set of break patterns. Such common sub-patterns have to be calculated and stored only once and can then be applied to different shifts.

We compute  $\hat{\mathcal{D}}_S$  for each shift as follows: If, according to constraint  $C_2$  (see Sect. 2), the shift contains a lunch break, we consider each timeslot the lunch break may be assigned to according to  $C_2$  and divide the shift into periods before and periods after each lunch break position. A set of break patterns for any of these periods depends on the length  $p$  of the period, the breaktime  $b$  it must contain, and  $\mathcal{C}$ . Since  $\mathcal{C}$  is defined globally, a set of break patterns can be identified by  $(p, b)$ . That means that any period of length  $p$ , in which  $b$  timeslots have to be assigned 0-slots, has the same set of valid break patterns. We compute a set of sub-patterns for each possible period before and after lunch breaks and for each period covering the shifts that do not contain a lunch break. Since periods with the same length and breaktime share the same set of patterns, we only have to compute a subset of patterns for each pair  $(p, b)$ . For each shift length we store only references to the sets of sub-patterns.

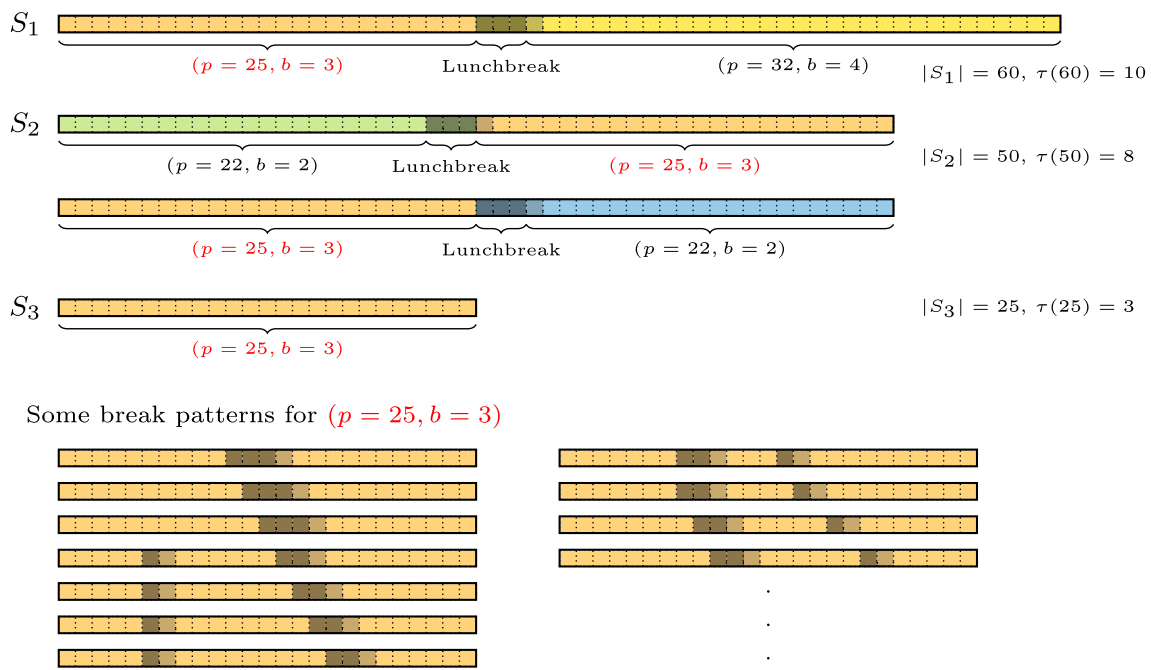


Fig. 5 Computation of break patterns

An example for this method is depicted in Fig. 5. The first shift,  $S_1$ , is divided into two parts by its lunch break. The first part consists of 25 timeslots and the second part of 32 timeslots. Note that a different location of the lunch break could be possible, but is not depicted here. We want to assign three 0-slots to the first part and four 0-slots to the second part. Again, a different assignment may be possible, such as assigning four 0-slots to the first part, and three to the second part, as long as none of the constraints in  $\mathcal{C}$  is violated and the number of 0-slots amounts to  $\tau(|S_1|)$ . For the second shift,  $S_2$  two possibilities for placing the lunchbreak are shown.

In shift  $S_3$  no lunch break is required. In all shifts a period with attributes  $(p = 25, b = 3)$  occurs. These periods share the same set of break patterns.

Whenever a fresh random break pattern is needed during the algorithm, we can just take one from the respective table and do not depend on the cubic time algorithm needed to generate a break pattern on-the-fly. These precalculated sets of break patterns are used in the initialisation of the algorithm and by the local search and mutation operators.

Table 1 shows the sizes of  $\widehat{\mathcal{D}}_S$  according to the constraints for different shift lengths in the publicly available problem

Table 1 Sizes of  $\widehat{\mathcal{D}}_S$  for different shift lengths from our real-life benchmarks

$ S $	$ \widehat{\mathcal{D}}_S $	$ S $	$ \widehat{\mathcal{D}}_S $
60	2,514	126	56,150,948
92	14,149	138	374,959,311
102	131,698	144	1,133,795,593

instances. Specific information on these instances is given in Sect. 7.

### 4.3 Generating the initial population

Each individual  $I$  in the population is initialised in two steps: First, for each shift  $S \in \mathcal{S}$  a valid break pattern  $D$  is selected randomly from  $\widehat{\mathcal{D}}_S$ . This provides us with a first solution satisfying the temporal constraints  $\mathcal{C}$ . Second, a simple local search using neighbourhood  $\mathcal{N}_1$ , as described below, is executed on the solution.

### 4.4 Neighbourhoods

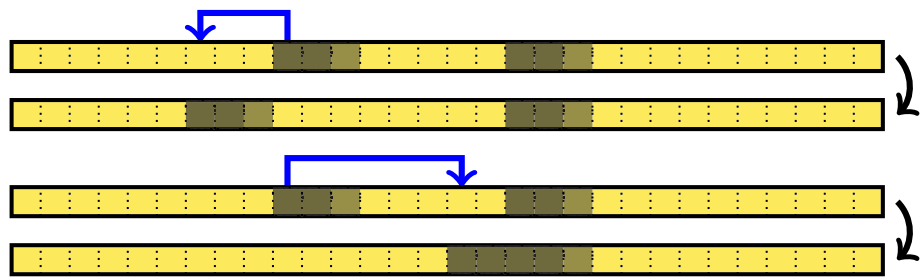
A neighbour of a solution  $\mathcal{B}$  of the BSP is another solution  $\mathcal{B}'$  which can be reached from  $\mathcal{B}$  by applying a move. A move is a small change on the solution resulting in another, better or worse, solution.

A neighbourhood consists of all possible solutions that can be obtained by a certain type of move. All our move types depend on a break  $B$ . We define three different neighbourhoods of very different size.

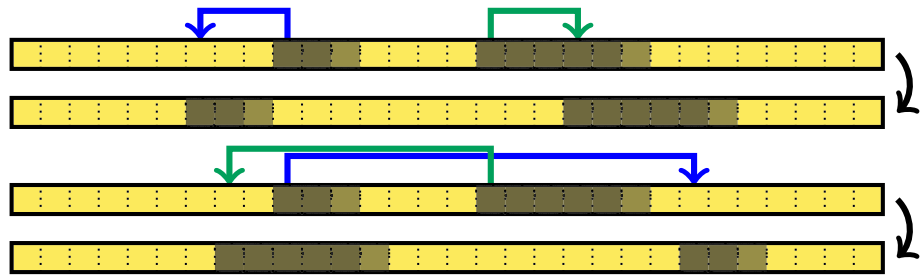
**Definition 23** (Single assignment neighbourhood  $\mathcal{N}_1$ ) The set of all solutions that are reached by moving a break  $B$  to a different set of timeslots under consideration of  $\mathcal{C}$ . This includes appending  $B$  to its predecessor or successor, which results in one longer break. Examples are depicted in Fig. 6. For the problem instances used in this work (for details see



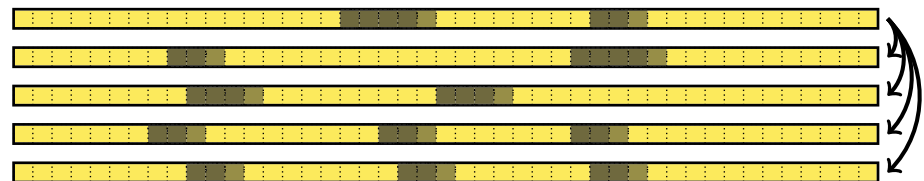
**Fig. 6** Two possible single assignment moves of a break within a shift



**Fig. 7** Two possible double assignment moves of a break within a shift



**Fig. 8** Four possible shift assignment moves exchanging the whole break pattern



Sect. 7), the size of this neighbourhood averages three to four neighbours.

**Definition 24** (*Double assignment neighbourhood  $\mathcal{N}_2$* ) The set of all solutions that are reached by moving a break  $B$  and its predecessor or successor to different sets of timeslots under consideration of  $\mathcal{C}$ . Like in  $\mathcal{N}_1$ , two breaks can be joined to form a longer break. Two breaks of different length can be swapped. This neighbourhood is significantly larger than  $\mathcal{N}_1$ . For the instances tested, its size amounts to up to 100 neighbours. This neighbourhood is illustrated in Fig. 7.

**Definition 25** (*Shift assignment neighbourhood  $\mathcal{N}_3$* ) The set of solutions that are reached by changing the whole break pattern of the shift that contains  $B$ . The possible patterns are retrieved from the pre-calculated set of break patterns  $\widehat{\mathcal{D}}_S$  described in Sect. 4.2. For performance reasons, we do not consider the complete  $\widehat{\mathcal{D}}_S$ , but only a randomly selected subset. The size of this subset determines the size of the neighbourhood. Figure 8 depicts this neighbourhood. This neighbourhood is likely to contribute to diversification rather than intensification. This behaviour is also reflected in the evaluations described in Sects. 5.4 and 6.4.

#### 4.5 Local search

At each iteration of the local search the following steps are performed on an individual  $I$ . First, a break  $B$  is selected at random out of a set  $\widehat{B}$  of breaks.  $\widehat{B}$  may comprise all

breaks that are currently included in the individual’s solution, or a subset thereof. Second, a neighbourhood out of the three neighbourhoods  $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$  is chosen according to the parameter  $\eta = (\eta_1, \eta_2, \eta_3)$  which represents the probability for each neighbourhood to be selected. Then the set  $\mathcal{N}$  of all neighbours according to the chosen neighbourhood is computed.

Next, for  $N \in \mathcal{N}$  let  $\delta(N, I) = F(N) - F(I)$ , i.e., the difference between the fitness value of an individual and its neighbours, and let  $\mathcal{N}' = \{N \in \mathcal{N} \mid \delta(N, I) \leq 0\}$ . If  $|\mathcal{N}'| > 0$  then  $I$  is assigned the  $N \in \mathcal{N}'$  for which  $\delta(N, I)$  is minimal, ties broken randomly. Thus,  $I$  is assigned its best neighbour. Otherwise nothing happens. The local search terminates when for  $\mu$  subsequent iterations  $|\mathcal{N}'| = 0$ , i.e., no neighbours with better or equal fitness could be found.

This procedure is influenced by three parameters: the size of  $B$ , the search intensity determined by  $\mu$ , and the probabilities  $\eta$  of the different neighbourhoods. Tests on different values for these parameters are described in Sects. 5.4 and 6.4. Algorithm 1 outlines the local search procedure.

### 5 MABS: memetic algorithm for break scheduling

This algorithm creates each offspring either by mutation or by crossover from the previous generation. A  $k$ -tournament selector [18] decides which individuals survive in each iteration. The local search is applied in each iteration on a subset

**Algorithm 1** LocalSearch (Individual  $I$ , Breaks  $B$ )

---

```

1:  $c \leftarrow 0$ 
2: repeat
3:    $B \leftarrow$  select break  $B \in \hat{B}$  randomly
4:    $\mathcal{N} \leftarrow$  select and compute one of  $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ 
5:    $\mathcal{N}' \leftarrow \{N \in \mathcal{N} \mid \delta(N, I) \leq 0\}$ 
6:   if  $|\mathcal{N}'| > 0$  then
7:      $I \leftarrow N \in \mathcal{N}'$  with minimal  $\delta(N, I)$ 
8:      $c \leftarrow 0$ 
9:   else
10:     $c \leftarrow c + 1$ 
11:   end if
12: until  $c == \mu$ 
13: return  $I$ 

```

---

of the population. Algorithm 2 outlines this method. We also experimented with the application of a tabu list in the local search procedure.

**Algorithm 2** MABS - Memetic Algorithm for Break Scheduling

---

```

1: BUILDBREAKPATTERNS
2:  $\mathcal{I} \leftarrow$  Initialisation
3: repeat
4:    $E \leftarrow$  fittest  $I \in \mathcal{I}$ 
5:   for all individuals  $I \in \mathcal{I} \setminus \{E\}$  do
6:      $x \leftarrow$  select random float uniformly distributed in  $[0..1]$ 
7:     if  $x \leq \alpha$  then
8:        $J \leftarrow$  select random individual  $J \neq I$ 
9:        $I \leftarrow$  CROSSOVER( $I, J$ )
10:    else
11:       $I \leftarrow$  MUTATE( $I$ )
12:    end if
13:   end for
14:    $\mathcal{I} \leftarrow$  SELECT( $\mathcal{I} \setminus \{E\} \cup \{E\}$ )
15:    $\mathcal{I}_F \leftarrow |\mathcal{I}| \cdot \lambda$  fittest individuals in  $\mathcal{I}$ 
16:   for all  $I \in \mathcal{I}_F$  do
17:      $\hat{B} \leftarrow$  all breaks contained in  $I$ 
18:      $I \leftarrow$  LOCALSEARCH( $I, \hat{B}$ )
19:   end for
20: until timeout
21: return fittest  $I \in \mathcal{I}$ 

```

---

## 5.1 Crossover and mutation

Each individual  $I \in \mathcal{I} \setminus \{E\}$  is replaced by an offspring created either by mutation or by crossover. Crossover takes place with probability  $\alpha$  and mutation with  $1 - \alpha$ . The crossover operator selects a partner  $J \in \mathcal{I}$  with  $J \neq I$  randomly out of the generation and creates an offspring inheriting each meme from either of the parents. The decision on which meme to inherit from which parent can be taken randomly or with a probability  $\gamma$  to inherit the meme  $M$  with better fitness. Values for the parameters  $\alpha$  and  $\gamma$  are evaluated in Sect. 5.4. The mutation operator performs one random move on the given individual using the *shift assignment* neighbourhood.

## 5.2 Selection

The selection operator selects a set of individuals that survive the current iteration. The elitist is excluded from the selection process; it survives without passing a selection. Among the other individuals, a  $k$ -tournament selector [18] is applied. This operator randomly takes  $k$  individuals,  $k \leq |\mathcal{I}|$ , out of the generation to perform a tournament. Out of these  $k$  individuals, the one with the best fitness value survives. This procedure is repeated  $|\mathcal{I}| - 1$  times. The population then consists of the elitist and the individuals selected by the tournament.

## 5.3 Local search with tabu list

MABS performs the local search as described earlier on a subset of individuals. The set  $\hat{B}$  of breaks, on which the local search is conducted, contains all breaks of the solution of the individual. In case the tabu list is used, the following modification applies to the search: the computed neighbourhood  $\mathcal{N}$  is reduced by those neighbours that currently reside in the tabu list. A tabu list [17]  $L$  is maintained for each break  $B$ .  $L$  contains the timeslots the first slot of  $B$  has previously been assigned to. Whenever a move is performed by  $B$ ,  $L$  is updated by deleting the oldest value in  $L$  and adding the current first slot of  $B$ . The length of the tabu list thus determines for how long a value is kept. If any of the tabu moves leads to a globally improved neighbour, however, it is allowed anyway. The tabu list intends to prevent the local search from re-visiting previously computed solutions.

## 5.4 Parameter evaluation

We evaluated a set of parameters that influence the quality of solutions. For the parameter evaluation we selected a set of six different instances among 30 problems presented by Beer et al. [6], which are publicly available in [27]. The timeout was set to 3,046 s according to a benchmark on the machine used by Beer et al. [6] and our machine. The algorithm was run ten times for each instance and parameter value. The impact of each parameter was assessed using the Kruskal–Wallis test [19]. Detailed results of this evaluation can be found in Sect. 5.2.6 of [30], where this algorithm is found under the name “MAR2”.

The following parameters were evaluated for this algorithm:

$ \mathcal{I} $	Population size. Values tested: 1, 4, 10, 20, 40, 70, best $ \mathcal{I}  = 4$ .
$\gamma$	Crossover: Probability to select the fitter meme. Values tested: 0.0, 0.6, 0.9, no significance.
$\alpha$	Crossover vs Mutation: Probability to create offspring by crossover, $1 - \alpha$ probability

	ity to create offspring by mutation. Values tested: 0.5, 0.7, 0.9, no significance.
$\kappa$	Selection pressure: Number of individuals performing a tournament. Values tested: 1, 2, 3, best $\kappa = 1$ .
$\lambda$	Search rate, percentage of population the local search is applied on. Values tested: 0.1, 0.2, 0.5, 0.8, best $\lambda = 0.8$ .
$ L $	Length of tabu list. Values tested: 0, 1, 2, 4, best $ L  = 0$ .
$\mu$	Search intensity: Number of iterations the local search continues without finding improvements. This value is multiplied by the number of breaks $ \widehat{B} $ available to the local search. Values tested: 2, 6, 8, 10, 15, best $\mu = 10$ .
$\eta = (\eta_1, \eta_2, \eta_3)$	Probability for each neighbourhood to be selected in each local search iteration. Values tested: (1, 0, 0), (0, 1, 0), (0, 0, 1), (.6, .3, .1), (.3, .6, .1), (.3, .1, .6), best $\eta = (.3, .6, .1)$ .

The population size made a significant difference in most of the instances, with the value performing best being  $|Z| = 4$ . This value being larger than 1 means that the genetic operators indeed have an impact the solution quality. At the same time, the values for two out of three parameters that influence the genetic operators gave no significant difference. The parameter  $\kappa$  defining the selection pressure did have an impact on the solution qualities, but the best value was  $\kappa = 1$ , i.e., the algorithm performed best when no selection pressure was applied at all. Other than the parameters for the genetic operators, most of the parameters for the local search, i.e.,  $\lambda$ ,  $|L|$ ,  $\mu$  and  $\eta$  significantly influenced the solution qualities. An interesting outcome is that the use of the tabu list actually worsened the solution qualities, as for most of the instances tested, the best results were obtained with a tabu list of 0 length.

A possible interpretation of these results is that the genetic operators mainly diversify solutions and the local search does most of the improvements.

The high value for  $\lambda$  supports this interpretation. A tabu list is usually applied to guide the algorithm away from local optima. A possible reason why the tabu list did not improve our results is that it is too restrictive and its combination with the genetic operators results in too much diversification.

### 5.5 Weaknesses of the algorithm

In addition to the parameter evaluation discussed above, an analysis of the log files of the evaluation runs allowed us to identify the following weaknesses of the algorithm.

1. The contribution of the genetic operators consisted mainly in diversification, but at least the crossover operator should be expected to contribute to an improvement of the solutions. Often good and diverse memes, i.e., memes with a good distribution of breaks, were deleted by the random selection of memes inside the crossover operator.
2. The local search was performed on whole solutions, i.e., on all memes of a solution, even though some memes had more potential to be improved than others.
3. The tabu list contained positions of breaks. It is possible that due to the very high number of combinations of break patterns in shifts containing the same timeslots, the tabu list blocked the way to new optima at a too early stage.

## 6 MAPBS: memetic algorithm with penalties for break scheduling

The unusual results of the parameter evaluation of MABS along with some intuition on possible weaknesses of this algorithm led us to a complete re-design, resulting in the new algorithm MABPS.

To avoid discarding too many good memes, as had been observed in MABS, we designed the crossover operator to work on memes of the whole generation rather than on two selected parents. This means that an offspring can have more than two parents and each individual is more likely to become a parent. In each iteration the best memes of the current meme-pool are put together into one individual to make sure they survive. We further included the selection mechanism into the crossover operator. The selection is now performed on memes rather than on individuals. This way, good memes, which may be part of a bad individual, are also less likely to be discarded.

The local search was changed to focus on memes that have more potential to be improved. To assess this potential, memes keep a memory to track their improvement history. We call this memory a *penalty system*. A high penalty value indicates a local optimum. Memes with a high penalty value are less likely to be searched further and more likely to be discarded by the crossover operator.

The mutation now acts as a diversification on single memes rather than on whole individuals.

Algorithm 3 outlines the procedure. In the following we describe the penalty system, the new crossover and selection mechanism, and the application of the local search and mutation in MAPBS.

### 6.1 Penalty system

For each meme instance  $M$  we additionally store the following values:

---

**Algorithm 3** MAPBS - Memetic Algorithm with Penalties for Break Scheduling
 

---

```

1: BUILDBREAKPATTERNS
2:  $\mathcal{I} \leftarrow$  Initialisation
3: repeat
4:    $\mathcal{I} \leftarrow$  CROSSOVER-SELECT( $\mathcal{I}$ )
5:    $E \leftarrow$  fittest  $I \in \mathcal{I}$ 
6:   for all individuals  $I \in \mathcal{I} \setminus \{E\}$  do
7:      $\mathcal{M}' \leftarrow$  elements of  $\mathcal{M}$  with lowest penalty value  $P_M$ 
8:      $I \leftarrow$  MUTATE( $I, \mathcal{M}'$ )
9:      $\widehat{B} \leftarrow$  all breaks contained in  $\mathcal{M}'$ 
10:     $I \leftarrow$  LOCALSEARCH( $I, \widehat{B}$ )
11:     $I \leftarrow$  PENALTY-UPDATE( $I, \widehat{B}$ )
12:   end for
13: until timeout
14: return fittest  $I \in \mathcal{I}$ 

```

---

- Best fitness value  $\bar{F}_M$ : The best value for  $F_M$  the meme ever reached
- Penalty value  $P_M$ : Number of iterations since last update of  $\bar{F}_M$

The higher  $P_M$ , the longer the meme was not able to improve. This means it is more likely to be stuck in a local optimum. We use this value in two parts of the algorithm: The crossover operator prefers memes with a low value for  $P_M$  to eliminate memes stuck in local optima, disregarding their fitness value  $F_M$ . Second, the subset of memes which is used for the mutation and local search also prefers memes with low  $P_M$  and thereby focuses on areas where improvements are more likely to be found. After each iteration, the values for  $\bar{F}_M$  and  $P_M$  are updated for each  $M \in \mathcal{M}$  as follows.

$$\begin{cases} \bar{F}_M = F_M, P_M = 0, & \text{if } \bar{F}_M < F_M \\ P_M = P_M + 1 & \text{otherwise} \end{cases}$$

### 6.2 Crossover and selection

First, an individual is created by selecting for each meme  $\bar{M}$  its instance  $M$  with the best current fitness value  $F_M$  out of the current meme-pool. This individual is likely to be the elitist in the current population. Second, each of  $|\mathcal{I}| - 1$  individuals is created as follows. For each meme  $\bar{M}$  we perform a  $k$ -tournament selection [18] on the set of its instances in the current meme-pool. We select  $k$  instances at random out of the current meme-pool and inherit the instance  $M$  with the lowest penalty value  $P_M$  to the offspring. The first part assures to survive the best meme instances of the current meme-pool. The second part forms the actual crossover procedure. By using  $P_M$  as the selection criterion, we get rid of meme instances that have been stuck in local optima for too long. If a local optimum constitutes a global optimum, then it survives through the first step of the crossover operator. Figure 9 depicts the crossover operator.

### 6.3 Mutation and local search

On each individual  $I \in \mathcal{I} \setminus \{E\}$  the following steps are performed: A set  $\mathcal{M}' \in \mathcal{M}$  of meme instances is selected such that  $\mathcal{M}'$  contains the meme instances with the lowest penalty values (ties are broken randomly). Each  $M' \in \mathcal{M}'$  is mutated as follows. A set of shifts  $\mathcal{S}' \in \mathcal{S}_{M'}$  is chosen at random. Then for each shift  $S' \in \mathcal{S}'$  its current break pattern is replaced by a pattern selected randomly out of the set  $\widehat{\mathcal{D}}_{S'}$  of break patterns computed in the beginning as described in Sect. 4.2. The size of  $\mathcal{S}'$  is a parameter value. Different values for this parameter are evaluated in Sect. 6.4. The local search is executed as described earlier with set  $\widehat{B}$  containing only the set of breaks contained in  $\mathcal{M}'$ .

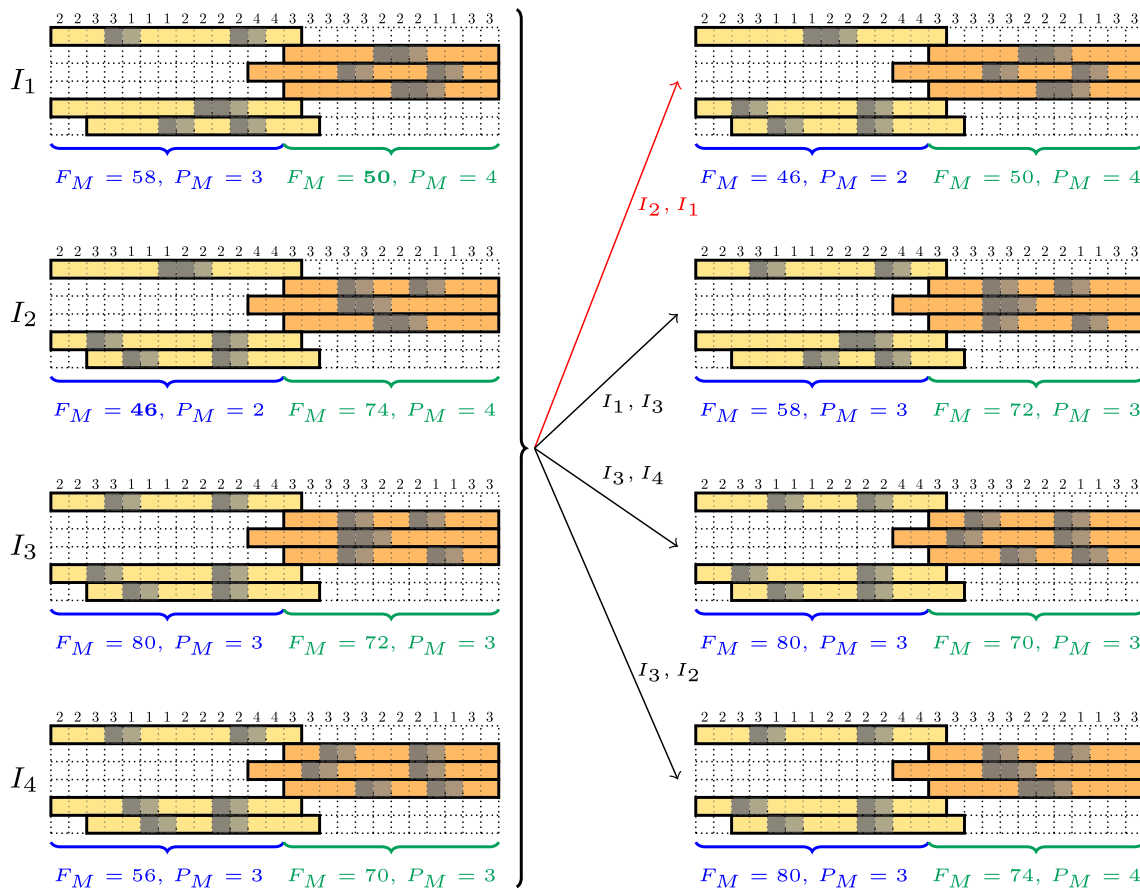
### 6.4 Parameter evaluation

As for MABS, we also evaluated parameters for MAPBS. The environment for this parameter evaluation was the same as for MABS in Sect. 5.4. Detailed results of this evaluation can be found in Sect. 5.2.7 of [30].

The following parameters were evaluated for this algorithm:

$ \mathcal{I} $	Population size. Values tested: 1, 4, 6, 10, 20, best $ \mathcal{I}  = 4$ .
$\lambda$	Percentage of meme instances in $\mathcal{M}$ being mutated and locally improved for each individual (size of $ \mathcal{M}' $ ). Values tested: 0.05, 0.1, 0.2, 0.3, 0.5, best $\lambda = 0.05$ .
$\sigma$	Mutation weight, percentage of shifts being mutated. Values tested: 0.01, 0.05, 0.1, 0.3, 0.5, best $\sigma = 0.05$ .
$\kappa$	Selection, number of memes performing a tournament in the crossover operator. Values tested: 1, 2, best $\kappa = 1$ .
$\mu$	Search intensity, number of iterations the local search continues without finding improvements. This value is multiplied by the number of breaks $ \widehat{B} $ available to the local search. Values tested: 10, 20, 30, 40, best $\mu = 20$ .
$(\eta_1, \eta_2, \eta_3)$	Probability for each neighbourhood to be selected in each local search iteration. Values tested: (0, .5, .5), (.5, 0, .5), (.5, .5, 0), (.2, .8, 0), (.8, .2, 0), (.3, .3, .3), (1, 0, 0), (0, 1, 0), best $\eta = (.8, .2, 0)$ .

This algorithm also performed best with a small population size. As for MABS, we also tested a population size of  $|\mathcal{I}| = 1$  to make sure that the population based approach is indeed necessary to obtain good solutions.



**Fig. 9** Crossover operator. The first offspring is created by choosing only the fittest instance of each meme, i.e.,  $M_1$  from  $I_2$  and  $M_2$  from  $I_1$ . The remaining offsprings are created by applying a  $k$ -tournament selection on each meme’s instances. The values for  $F_M$  are calculated

according to Definitions 13 and 17 with weights  $w_o = 2$  and  $w_u = 10$ . Different values for  $F_M$  after the crossover may occur from shifts overlapping into different memes

Since mutation may worsen a solution during the progress of the algorithm, for  $|\mathcal{I}| = 1$  the best obtained solution was kept in memory.

The mutation and search rate  $\lambda$  determining  $|\mathcal{M}'|$  led to the best results when kept low. On many problem instances,  $\lambda = 0.05$  leads to only one meme instance being mutated and searched. The mutation weight  $\sigma$  also worked well with a low value.  $\sigma$  determines the percentage of shifts which are assigned a new break pattern during a mutation. Similar to the other algorithm, the value of  $\kappa$  did not have an impact.

As in MABS, the local search intensity  $\mu$  was set relative to the number of breaks  $|\widehat{B}|$  taking part in the search. For this algorithm, larger values for  $\mu$  probably perform better than for MABS because  $|\widehat{B}|$  is much smaller. In MABS,  $\widehat{B}$  contains the complete set of breaks in the solution, but MAPBS considers only a subset of all breaks, namely those contained in  $\mathcal{M}'$ , which, according to the low value for  $\lambda$  were only a small subset. We tested some more neighbourhood combinations than for MABS. All runs where  $\mathcal{N}_3$  participated gave worse results than those where we used only

$\mathcal{N}_1$  and  $\mathcal{N}_2$ . The best performing combination was  $\eta_1 = 0.8$  and  $\eta_2 = 0.2$ . A possible reason for this result is that the shift assignment neighbourhood diversifies rather than intensifies the search. However, the algorithm is designed to diversify by the genetic operators rather than by the local search.

### 7 Comparison of results

For the parameter evaluation we selected a set of six different instances among 30 problems presented by Beer et al. [6], which are publicly available in [27]. 20 of them were retrieved from a real-life application and ten of them were generated randomly. The set of instances is the same as the one used by the authors of [6]. Details regarding the random generation are provided by the same authors in [27].

The input data  $\mathcal{C}$  (constraints) and  $k$  (number of timeslots) is the same for all random and real-life instances with  $k = 2, 016$  and  $\mathcal{C}$  defined as follows:

**Table 2** Comparison of results returned by the state-of-the-art algorithm [6], MABS, and MAPBS for real-life instances and random instances. The columns represent the best and average values, and the

standard deviation. The best results for the best and average values are highlighted in bold font

Instances	Beer et al. [6]			MABS			MAPBS		
	Best	Avg	$\sigma$	Best	Avg	$\sigma$	Best	Avg	$\sigma$
2fc04a	3,094	3,224	84	3,244	3,326	50	<b>2,816</b>	<b>2,961</b>	71
2fc04a03	3,100	3,200	61	3,220	3,328	57	<b>2,834</b>	<b>2,934</b>	54
2fc04a04	3,232	3,342	68	3,226	3,297	44	<b>2,884</b>	<b>2,954</b>	60
2fc04b	<b>1,822</b>	2,043	92	2,266	2,387	68	1,884	<b>1,948</b>	49
3fc04a	1,644	1,767	102	1,810	1,909	59	<b>1,430</b>	<b>1,533</b>	67
3fc04a03	1,632	1,759	87	1,846	1,944	55	<b>1,440</b>	<b>1,514</b>	40
3fc04a04	1,932	1,980	40	1,930	2,056	87	<b>1,614</b>	<b>1,718</b>	48
3si2ji2	3,626	3,667	35	3,344	3,398	27	<b>3,177</b>	<b>3,206</b>	17
4fc04a	1,694	1,817	126	1,814	1,972	139	<b>1,478</b>	<b>1,540</b>	29
4fc04a03	1,666	1,795	87	1,742	1,870	59	<b>1,430</b>	<b>1,502</b>	42
4fc04a04	1,918	2,017	95	1,850	1,980	60	<b>1,606</b>	<b>1,674</b>	48
4fc04b	1,410	1,489	49	1,628	1,772	154	<b>1,162</b>	<b>1,233</b>	48
50fc04a	1,672	1,827	81	2,018	2,090	32	<b>1,548</b>	<b>1,603</b>	36
50fc04a03	1,686	1,813	84	1,822	1,951	87	<b>1,402</b>	<b>1,514</b>	67
50fc04a04	1,790	1,917	64	1,914	2,009	48	<b>1,480</b>	<b>1,623</b>	89
50fc04b	1,822	2,012	91	2,322	2,464	98	<b>1,818</b>	<b>1,900</b>	56
51fc04a	2,048	2,166	89	2,490	2,836	687	<b>1,886</b>	<b>2,074</b>	87
51fc04a03	1,950	2,050	86	2,318	2,377	37	<b>1,886</b>	<b>1,949</b>	46
51fc04a04	2,058	2,191	64	2,370	2,728	678	<b>1,958</b>	<b>2,039</b>	52
51fc04b	<b>2,244</b>	2,389	94	2,796	2,950	88	2,306	<b>2,367</b>	43
random1-1	728	972	177	544	592	41	<b>346</b>	<b>440</b>	48
random1-2	1,654	1,994	172	712	817	92	<b>370</b>	<b>476</b>	65
random1-5	1,284	1,477	99	696	742	47	<b>378</b>	<b>418</b>	29
random1-7	860	1,077	154	824	940	73	<b>496</b>	<b>583</b>	42
random1-9	1,358	1,658	213	672	734	38	<b>318</b>	<b>423</b>	51
random1-13	1,264	1,535	245	570	699	68	<b>370</b>	<b>445</b>	55
random1-24	1,586	1,713	74	884	934	46	<b>542</b>	<b>611</b>	43
random1-28	1,710	2,020	233	626	726	71	<b>222</b>	<b>318</b>	71
random2-1	1,686	1,855	142	914	1,058	91	<b>724</b>	<b>889</b>	75
random2-4	1,712	2,053	242	794	889	56	<b>476</b>	<b>535</b>	45

- $C_1$  Break positions:  $d_1 = d_2 = 6$ .
- $C_2$  Lunch breaks:  $h = 72, g = 6, l_1 = 42, l_2 = 72$ .
- $C_3$  Duration of work periods:  $w_1 = 6, w_2 = 20$ .
- $C_4$  Minimum break times:  $w = 10, b = 4$ .
- $C_5$  Break durations:  $b_1 = 2, b_2 = 12$ .

All values are given in timeslots with one timeslot corresponding to 5 min.  $k$  thus represents one calendar week.

The real-life instances were drawn from a real-life problem in the area of supervision personnel. They are characterised by two main factors: Different staffing requirements and different forecast methods. Staffing requirements vary according to calendar weeks. A forecast method is a specific way of planning a future shiftplan, influencing the num-

ber of shifts and the shift lengths. As shown in Sect. 4.2, the domain size  $\mathcal{D}_S$  grows exponentially with respect to the shift length  $|S|$ . Therefore, instances with short shifts have a smaller search space.

As for the parameter evaluation, the timeout per run was normalised to 3,046 s according to a benchmark of the machine of [6] and ours. This allows us a more reliable comparison of our results to the best existing upper bounds for the BSP presented in [6]. We ran the algorithm ten times for each instance and parameter value. Each run was performed on one core with 2.33 GHz of a QuadCore Intel Xeon 5345 with three runs being executed simultaneously, i.e., three cores being fully loaded. The machine provides 48 GB of memory. A more detailed description of the empirical parameter evaluation can be found in [30].

For the final runs we used the following settings for MABS:  $|Z| = 4$ ,  $\gamma = 0.9$ ,  $\kappa = 1$ ,  $\alpha = 0.9$ ,  $\lambda = 0.8$ ,  $|L| = 0$ ,  $\mu = 10$  and  $\eta = (0.3, 0.6, 0.1)$ , and for MAPBS:  $|Z| = 4$ ,  $\lambda = 0.05$ ,  $\sigma = 0.05$ ,  $\mu = 20$ ,  $\kappa = 1$  and  $\eta = (0.8, 0.2, 0.0)$ . Table 2 compares the results of MABS and MAPBS to state-of-the-art results from [6]. The numbers in columns *Best* and *Avg* represent the best and average values of the objective function from Definition 13 over ten runs.

The algorithm presented in [6] has shown to be very good in practice and has been used in real-life applications. Based on Table 2 we can conclude that our MABS algorithm manages to improve state-of-the-art results for random instances, but it is outperformed by [6] in most real-life instances. Our second algorithm (MAPBS) returns improved results on all random instances and 18 out of 20 real-life instances compared to both results from literature and results returned by MABS. Another feature of MAPBS is the low standard deviation  $\sigma$  which makes the algorithm more reliable.

## 8 Conclusions and future work

We proposed two different memetic approaches to optimise BSP. In both algorithms we applied a local search heuristic for which we proposed three different neighbourhoods. Further, we introduced a method to avoid local optima based on penalties for parts of solutions which are not improved during a number of iterations.

We justified the choice of a metaheuristic to optimise BSP by presenting an NP-completeness proof for BSP under the condition that the input contains break patterns explicitly.

For each algorithm we conducted a set of experiments with different parameter settings and then compared the algorithms with their best settings. The following conclusions can be drawn from the parameter evaluation:

- Using genetic operators combined with a local search returns better results than using only a local search.
- Applying the local search either only on some individuals or only on small parts of each individual's solution significantly improves the qualities of the solutions compared to applying the local search on all individuals and entire solutions.
- Using the two smaller neighbourhoods in the local search returns better solutions than using only one neighbourhood. The largest of the proposed neighbourhoods performs worst.
- The use of a penalty system to guide the local search towards meme instances that are not likely to be in local optima significantly improves the quality of the solutions.

We compared our results to the best existing results in literature for 30 publicly available benchmarks. Our algorithm

returned improved results for 28 out of 30 instances. To the best of our knowledge, our solutions represent the new upper bounds for the available real-life problems.

For future work, it would be interesting to see how the MAPBS algorithm performs on long runs. The small population resulting from the parameter evaluation in Sect. 6.4 could be due to a short runtime.

Another interesting question is how to solve the two problems of shift scheduling and break scheduling in a single phase by a memetic approach. Dealing with the entire problem when the number of breaks for each shift is large is a challenging task due to a much larger search space.

**Acknowledgments** This work was supported by the Austrian Science Fund (FWF) under grants P24814-N23 and S11409-N23, and by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018.

## References

1. Aykin T (1996) Optimal shift scheduling with multiple break windows. *Manag Sci* 42:591–603
2. Aykin T (2000) A comparative evaluation of modelling approaches to the labour shift scheduling problem. *Eur J Oper Res* 125:381–397
3. Bechtold S, Jacobs L (1990) Implicit modelling of flexible break assignments in optimal shift scheduling. *Manag Sci* 36(11):1339–1351
4. Beer A, Gaertner J, Musliu N, Schafhauser W, Slany W (2008) An iterated local search algorithm for a real-life break scheduling problem. In: *Proceedings of Matheuristics 2008, 2nd international workshop on model based Metaheuristics*, Bertinoro
5. Beerm A, Gaertner J, Musliu N, Schafhauser W, Slany W (2008) Scheduling breaks in shift plans for call centers. In: *Proceedings of the 7th international conference on the practice and theory of automated timetabling*, Montreal
6. Beer A, Gärtner J, Musliu N, Schafhauser W, Slany W (2010) An AI-based break-scheduling system for supervisory personnel. *IEEE Intell Syst* 25(2):60–73
7. Burke EK, Cowling PI, Causmaecker PD, Berghe GV (2001) A memetic approach to the nurse rostering problem. *Appl Intell* 15(3):199–214
8. Côté M-C, Gendron B, Quimper C-G, Rousseau L-M (2011) Formal languages for integer programming modeling of shift scheduling problems. *Constraints* 16(1):55–76
9. Côté M-C, Gendron B, Rousseau L-M (2011) Grammar-based integer programming models for multiactivity shift scheduling. *Manag Sci* 57(1):151–163
10. Cotta C, Fernández AJ (2007) Memetic algorithms in planning, scheduling, and timetabling. *Evolutionary scheduling*. Springer, Berlin
11. Dantzig GB (1954) A comment on Eddie's traffic delays at toll booths. *Oper Res* 2:339–341
12. Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. *Artif Intell* 49:61–95
13. Di Gaspero L, Gärtner J, Kortsarz G, Musliu N, Schaerf A, Slany W (2007) The minimum shift design problem. *Ann Oper Res* 155:79–105
14. Garey M, Johnson D (1979) *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, New York

15. Gärtner J, Musliu N, Slany W (2004) A heuristic based system for generation of shifts with breaks. In: Proceedings of the 24th SGAI international conference on innovative techniques and applications of artificial intelligence, Cambridge
16. Gaspero LD, Gärtner J, Musliu N, Schaerf A, Schafhauser W, Slany W (2010) A hybrid LS-CP solver for the shifts and breaks design problem. In: Proceedings of the 7th international workshop on hybrid metaheuristics. Springer, Berlin/Heidelberg, pp 46–61
17. Glover F, Laguna M (1999) Tabu search. Handbook of combinatorial optimization, 3rd edn. Kluwer Academic Publishers, London
18. Goldberg DE, Deb K (1990) A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of genetic algorithms (FOGA). Morgan Kaufmann, San Francisco, pp 69–93
19. Montgomery D (2005) Design and analysis of experiments. Wiley, New York
20. Moscato P (1989) On evolution, search, optimization, gas and martial arts: towards memetic algorithms. In: Technical report of Caltech concurrent computer programming report, vol 826. California Institute of Technology, Pasadena
21. Musliu N, Schaerf A, Slany W (2004) Local search for shift design. Eur J Oper Res 153(1):51–64
22. Musliu N, Schafhauser W, Widl M (2009) A memetic algorithm for a break scheduling problem. In: 8th Metaheuristic international conference, Hamburg
23. Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization: algorithms and complexity. Prentice Hall, London
24. Quimper C-G, Rousseau L-M (2010) A large neighbourhood search approach to the multi-activity shift scheduling problem. J Heuristics 16(3):373–391
25. Rekik M, Cordeau J, Soumis F (2010) Implicit shift scheduling with multiple breaks and work stretch duration restrictions. J Sched 13:49–75
26. Schafhauser W (2010) TEMPLE—a domain specific language for modeling and solving real-life staff scheduling problems. PhD thesis, Vienna University of Technology, Wien
27. Softnet (2008) <http://www.dbai.tuwien.ac.at/proj/SoftNet/SuperVision/Benchmarks/>. Accessed 14 March 2014
28. Tellier P, White G (2006) Generating personnel schedules in an industrial setting using a tabu search algorithm. In: Burke EK, Rudova H (eds) The 5th international conference on the practice and theory of automated timetabling, pp 293–302
29. Thompson G (1995) Improved implicit modeling of the labor shift scheduling problem. Manag Sci 41(4):595–607
30. Widl M (2010) Memetic algorithms for break scheduling. Master's thesis, Vienna University of Technology, Vienna <http://www.kr.tuwien.ac.at/staff/widl/publications/Masterthesis.pdf>. Accessed 14 March 2014
31. Widl M, Musliu N (2010) An improved memetic algorithm for break scheduling. In: Hybrid Metaheuristics, vol 6373 of LNCS, pp 133–147