An Improved Memetic Algorithm for Break Scheduling

Magdalena Widl and Nysret Musliu

Institute of Information Systems, Vienna University of Technology, Austria
{widl, musliu}@dbai.tuwien.ac.at

Abstract. In this paper we consider solving a complex real life break scheduling problem. This problem of high practical relevance arises in many working areas, e.g. in air traffic control and other fields where supervision personnel is working. The objective is to assign breaks to employees such that various constraints reflecting legal demands or ergonomic criteria are satisfied and staffing requirement violations are minimised.
In our previous work we proposed a memetic algorithm for the assignment of breaks. We improve in this paper the previous method by proposing a new memetic representation, a new crossover and selection operator, and a penalty system that helps to select memes that have a better chance to be improved by a local search. Our approach is influenced by various parameters, for which we experimentally evaluate different settings. The impact of each parameter is statistically assessed. We compare our algorithm to state of the art results on a set of existing real life and randomly generated instances. Our new algorithm returns improved results on 28 out of the 30 benchmark instances. To the best of our knowledge, these results constitute current upper bounds for the respective instances.

1 Introduction

Breaks are periods during work shifts where staff is allowed, or in some cases obliged, to discontinue work in order to recover and to perform personal activities like having meals or using facilities. In many countries constraints for work and break periods are governed by federal law. Some employers might grant additional or extended breaks to comply with ergonomic needs of staff members and in some working areas breaks after certain working periods might even be crucial due to security related issues. While each employee is supposed to take breaks according to the mentioned constraints, also staffing requirements are to be fullfilled at all time, i.e. enough staff must be available to perform a specific task during a given timeslot.

Consider, for instance, airport security staff in charge of monitoring baggage x-ray machines: The person working in front of the monitor is required to keep high concentration in order to prevent mistakes that might result in hazardous items passing through. Thus, for all staff, breaks are mandatory to properly

recover after given periods of working time. Additionally, suppose there are estimated staffing requirements according to scheduled aircraft take-offs. Now breaks are to be scheduled such that all employees take breaks within given intervals, but at the same time a minimum number of employees is monitoring the screens. Instead of a minimum, we might even consider an exact number of employees to be present in order to minimize personnel costs.

Our particular problem statement origins from a real world scenario in the area of supervision personnel. We regard a shiftplan that consists of consecutive timeslots and of shifts. Each shift starts and ends in a specific timeslot and must contain a given amount of breaktime. Shifts may overlap in time. There are several constraints concerning the distribution of breaktime within a single shift such as minimum and maximum values limiting the length of breaks and worktime, to which we will refer as *temporal constraints*. Additionally, during each timeslot a given number of staff is required to be working. The breaktime for each shift is to be scheduled such that the temporal constraints are satisfied and staffing requirement violations are minimised.

We denote our formulation as Break Scheduling Problem and abbreviate it with Bsp.

In literature, the break assignment problem has mainly been addressed as part of the so-called shift scheduling problem, where shifts are scheduled along with breaks (see [2], [1], [6], [17], [15], [7], and [14]). However, these approaches consider up to 3-4 breaks per shifts while our problem definition does not restrict the number of breaks (the instances tested contain up to 11 breaks) and imposes several additional constraints. Recently, Di Gaspero et al [9] proposed a hybrid LS-CP solver for solving simultaneously the shift design and Bsp. This work gives promising results, but the best results for benchmark problems are still obtained by solving shift design and Bsp in two phases with a help of an expert. Bsp has been previously investigated in [4] and [13]. A similar problem for call centers, which additionally considers meetings and slightly different constraints, has been investigated in [3]. Beer et al. [4] applied a min-conflict based heuristic to solve the Bsp. We note that scheduling breaks considering only temporal constraints and leaving aside the staffing requirements can be formulated as simple temporal problem (STP) [8], which is solvable in polynomial time. In practical applications however the consideration of staffing requirements is crucial.

The main goal of this paper is to improve the memetic algorithm we proposed in [13]. Our current algorithm is based on the concept of memetic algorithms introduced by Moscato [12]. Memetic algorithms are also known as Hybrid Genetic Algorithms as presented by Goldberg [10]. The idea is to imitate cultural evolution on a pool of different solutions for an instance of an optimisation problem in order to obtain improved solutions. In contrast to purely genetic algorithms, local improvements are integrated in addition to the standard operators of biological evolution. It can thus be seen as a hybridisation of genetic operators with a local improvement method.

To solve Bsp we propose a new memetic representation, new crossover and selection mechanisms, and a penalty system to avoid local optima. We experi-

mentally evaluate the parameters that influence the optimisation process. The impact of each parameter is statistically verified. We finally compare our new memetic algorithm with the best existing results for this problem in the literature.

2 Problem Definition

Definition 1 (Timeslot t). *A time period of fixed length. In our real life problem, one timeslot corresponds to a period of five minutes.*

Definition 2 (Shift S). *A set of consecutive timeslots $S = \{t_i, t_{i+1}, ..., t_{i+n}\}$, $\forall j(i \leq j < i + n) : t_{j+1} - t_j = 1$. $S_s = t_i$ denotes the shift start and $S_e = t_{i+n}$ denotes the shift end. Each shift represents exactly one employee on duty.*

Definition 3 (Slot). *A timeslot in a particular shift. A slot can be either a break slot, a work slot or a re-acquaintance slot. The latter are those and only those slots that directly follow a sequence of break slots and stand for an employee getting familiar with an altered working situation after a break. During such a slot, the employee is not consuming breaktime but neither counted as working staff regarding staffing requirements.*

Definition 4 (Break B). *A set of consecutive break slots within a particular shift. The first slot in the set is referred to as break start, and the last slot as break end. A break is associated to exactly one shift.*

Definition 5 (Work period W). *A set of consecutive work slots and the subsequent re-acquaintance slot.*

Definition 6 (Breaktime). *The number of required break slots in a shift S. The breaktime depends on the shift's length $|S|$ and is given as input by a function $\tau(|S|)$.*

Definition 7 (Temporal constraints). *A set of global restrictions regarding lengths and locations of breaks and worktime within shifts. The following set of temporal constraints is part of our problem formulation:*

- C_1 *Break positions, defined by (d_1, d_2). The first d_1 and the last d_2 slots of each shift must be work slots, 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, defined by (h, g, l_1, l_2). Each shift S with $|S| > h$ must contain a lunch break L with $|L| \leq g$. The break may start earliest l_1 timeslots and end latest l_2 timeslots after the start of its associated shift.*
- C_3 *Work periods, defined by (w_1, w_2). For each work period W, $w_1 \leq |W| \leq w_2$.*
- C_4 *Minimum break duration, defined by (w, b). A work period W with $|W| \geq w$ must be followed by a break B with $|B| \geq b$.*
- C_5 *Break lengths, defined by (b_1, b_2). For each break B, $b_1 \leq |B| \leq b_2$.*

Definition 8 (Break pattern D_S). *A set $D_S \subset S$ of timeslots defining an assignment of breaks for a shift S satisfying breaktime $\tau(|S|)$ ($|D_S| = \tau(|S|)$) and the set of constraints $\mathcal{C}$.*

Definition 9 (Staffing requirements). *Number of required work slots for each timeslot $t \in T$.*

BSP is formally defined as follows:

Instance A tuple $(k, \mathcal{S}, \tau, \rho, \mathcal{C})$:

k: Number of timeslots defining a set of consecutive timeslots $T = \{t_1, t_2, ..., t_k\}$.
$\mathcal{S}$: Collection of shifts, each shift taking place within T, $\forall S \in \mathcal{S} : S \subseteq T$.
$\tau(|S|)$: Function mapping each shift length to a value denoting its breaktime.
$\rho(t)$: Function mapping each timeslot t to its staffing requirements.
$\mathcal{C}$: Set of temporal constraints $\{C_1, C_2, ..C_5\}$, as defined above.

A sample instance of BSP with a possible solution is depicted in Fig. 1 for instance $(k, \mathcal{S}, \tau, \rho, \mathcal{C})$ with $k = 30$, $\mathcal{S} = \{S_1, S_2, ..S_7\}$, $\tau(|S|) = 3$ if $|S| \leq 15; 4$ otherwise, ρ as stated in the second line, $C_1 = (3, 3)$, $C_2 = (25, 4, 7, 7)$, $C_3 = (3, 6)$, $C_4 = (5, 2)$, $C_5 = (1, 3)$

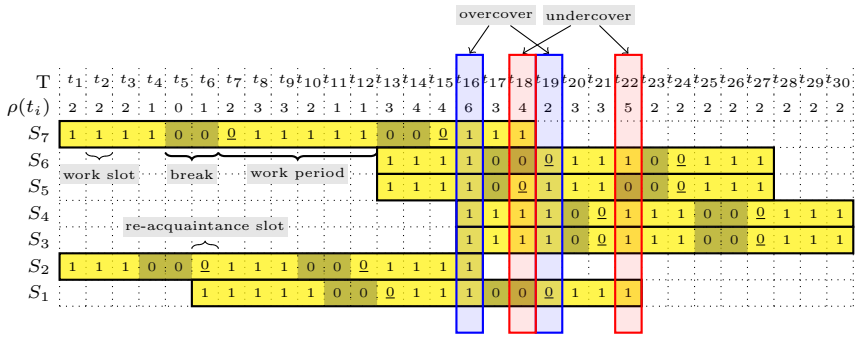

Fig. 1. A sample instance of BSP with solution

Objective Let $\mathcal{P} = (k, \mathcal{S}, \tau, \rho, \mathcal{C})$ be an instance of BSP. The objective is to find a mapping $\mathcal{B}$ relating each shift $S \in \mathcal{S}$ to a break pattern D_S, s.t. the following objective function is minimised:

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

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

3 A New Memetic Algorithm for the Break Scheduling Problem (MAPBS)

We propose a new memetic algorithm that differs in several aspects from the algorithm in [13]. This new algorithm uses a memetic representation that regards shifts which have many timeslots in common as memes. The new crossover operator uses memes of the whole generation to create an offspring instead of only two parents. The selection mechanism is included in the crossover operator. The mutation and local search procedures consider only breaks contained in a subset of an individual's memes. Memes keep a memory to track data about their search history. The new algorithm is abbreviated MAPBS for Memetic Algorithm with Penalty System for Break Scheduling. Algorithm 1 outlines the new proposed method with the components explained in the following.

Algorithm 1 Memetic Algorithm with Penalty System

```
1: buildBreakPatterns
2: I ← initialise
3: repeat
4:     E ← fittest I ∈ I
5:     for all individuals I ∈ I \ E do
6:         I ← penalty-update(I)
7:         I ← crossover-select(I)
8:         E ← fittest I ∈ I
9:         M' ← fittest M' ⊂ M
10:        I ← mutate(I, M')
11:        B ← all breaks contained in M'
12:        I ← search(I, B)
13:    end for
14: until timeout
15: return fittest I ∈ I
```

3.1 A new memetic representation

In the previous work [13] a memetic algorithm for Bsp was introduced. In this algorithm a meme M was represented by exactly one shift $S \in \mathcal{S}$ and its associated breaks according to $\mathcal{B}$, as depicted in Fig. 2.

This representation implies a strong interference between memes regarding the satisfaction of staffing requirements, which makes the design of effective genetic operators, especially crossover operators, difficult. Although this algorithm could give competitive results to the literature for the random instances, we further investigated other representations and added other features to the algorithm to improve results in literature.

We propose a new memetic representation that overcomes the problem of the previous representation by regarding sets of interfering shifts as memes. Shifts

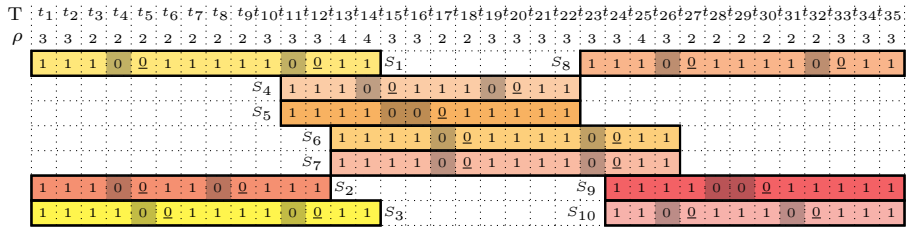

Fig. 2. Previous memetic representation

interfere if they have the bigger part of their timeslots in common and thus may assign breaks to the same timeslots.

Definition 10 (Memetic representation). *A meme $M \in \mathcal{M}$ is defined by a period $[m', m'')$, with $(m', m'') \in T^2$ and contains those and only those shifts $S \in \mathcal{S}$ where $m' \leq \lfloor (S_e + S_s)/2 \rfloor \leq m''$, S_s and S_e denoting shift start and end, as well as the breaks associated to these shifts. Each shift $S \in \mathcal{S}$ is thus contained in exactly one meme: $\forall M_i, M_j \in \mathcal{M} : M_i \cap M_j = \emptyset$, $M_1 \cup M_2 \cup \cdots \cup M_m = \mathcal{S}$.*

We use the following heuristic to determine the periods that induce the set of memes: For each $t \in T$ let a set of shifts $\mathcal{S}_t \subset \mathcal{S}$ s.t. $\forall S : S \in S_t$ iff $t \in S$, i.e. the set of shifts taking place during t. We assign a value $p(t)$ to each $t \in T$:

$$p(t) = \sum_{S \in \mathcal{S}_t} \begin{cases} 0 & \text{if } t < S_s + d_1 \\ 0 & \text{if } t > S_e - d_2 \\ 1 & \text{if } S_s + d_1 < t < S_s + d_1 + b_1 + w_1 \\ 1 & \text{if } S_e - d_2 > t > S_e - d_2 - b_1 - w_1 \\ 10 & \text{otherwise} \end{cases}$$

.

Recall that S_s and S_e denote start and end of shift S, d_1, d_2 denote the number of timeslots after S_s and before S_e respectively, to which no 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, as described in Section 2.

$p(t)$ serves as an indicator for the number of breaks that can be assigned to t. A low value for $p(t)$ tells us that there will be little intereferences between breaks regarding the staffing requirements, usually because a small number of shifts is taking place or many of them are ending or starting. This makes t a good point to separate one meme from another.

We determine a set of timeslots $T' \subset T$ with size $|T'|$ given by a parameter s.t.

- $\sum_{t' \in T'} p(t')$ is minimised
- $\forall t'_i, t'_j \in T' : \left|t'_i - t'_j\right| > d$ with $d = \lfloor (\min_{S \in \mathcal{S}} |S|)/2 \rfloor$, i.e. the distance between each pair of timeslots is at least half of the smallest shift length

To retrieve this set, we start with adding timeslot t'_0 to T' with $t'_0 = t$, $t \in T$ with minimal value $p(t)$, ties broken randomly, and continue by adding timeslots $t'_i = t$, $t \in T \setminus T'$ with minimal value $p(t)$ and $\forall t'_k, t'_j \in T' \cup t : \left|t'_k - t'_j\right| > d$. We obtain a set $\mathcal{M}$ of memes with $|\mathcal{M}| = |T'| - 1$ by sorting the elements in T' and defining each meme M_i by period $[t'_i, t'_{i+1})$.

Fig. 3 depicts the new memetic representation on a solution represented by a set of memes $\mathcal{M} = \{M_1, M_2, M_3\}$, $M_1 = (\{S_1, S_2, S_3\}, m' = 1, m'' = 12)$, $M_2 = (\{S_4, S_5, S_6, S_7\}, m' = 13, m'' = 23)$ and $M_3 = (\{S_8, S_9, S_{10}\}, m' = 24, m'' = 35)$.

Definition 11 (Meme fitness). *We define the fitness $F(M)$ of a meme M as the weighted sum of staffing requirement violations in all timeslots that are covered by the shifts contained in M, or more formally: Let $T' = \bigcup S, \forall S \in M$, then $F(M) = F(\mathcal{B}, \mathcal{P}, T') = w_o \cdot O(\mathcal{B}, \mathcal{P}, T') + w_u \cdot U(\mathcal{B}, \mathcal{P}, T')$*

- *w_o and w_u the same weights for over- and undercover defined in Section 2.*
- *$O(\mathcal{B}, \mathcal{P}, T') = \sum_{t \in T'} \max(0, \omega(\mathcal{B}, t) - \rho(t))$ i.e. sum of overcover*
- *$U(\mathcal{B}, \mathcal{P}, T') = \sum_{t \in T'} \max(0, \rho(t) - \omega(\mathcal{B}, t))$ i.e. sum of undercover*
- *$\omega(\mathcal{B}, t)$ the number of work slots in timeslot $t \in T'$ according to $\mathcal{B}$*

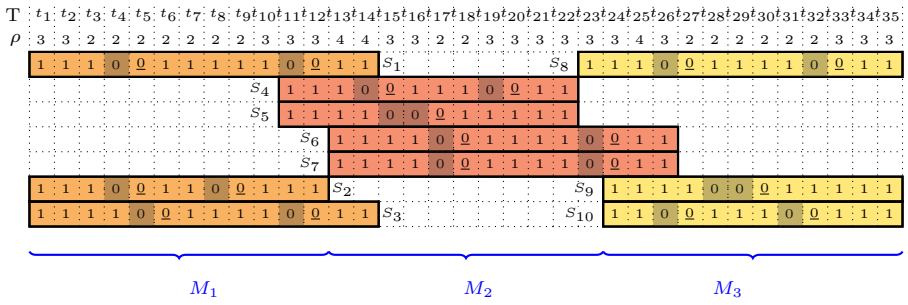

Fig. 3. New memetic representation

Definition 12 (Individual). *A solution $\mathcal{B}$ for an instance $\mathcal{P}$ and fitness value $F(I)$, which is the value of the objective function $F(\mathcal{B}, \mathcal{P})$.*

Definition 13 (Population). *A set $\mathcal{I}$ of individuals.*

Definition 14 (Generation). *Population during an iteration of the algorithm.*

Definition 15 (Memepool). *The set of all memes in a generation.*

Definition 16 (Elitist). *The individual $E \in \mathcal{I}$ with the best fitness value.*

3.2 Initialisation

We first precalculate a set $\mathcal{D}'_S$ of break patterns for each shift S using the small temporal problem model presented by [8]. Details on this procedure are given in [18].

Each individual in the population is initialised in two steps: First, for each shift $S \in \mathcal{S}$ a valid break pattern $D \in \mathcal{D}'_S$ is selected randomly. 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 in 3.5, is executed on the solution.

3.3 Penalty system

For each meme M we keep the following values:

Best fitness value $B(M)$**:** The best value for $F(M)$ the meme ever reached
Penalty value $P(M)$**:** Number of iterations since last update of $B(M)$

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 optima. We use this value at two points of the algorithm: The crossover operator prefers memes with low $P(M)$, thus memes stuck in local optima are likely to be eliminated, disregarding their fitness value $F(M)$. Second, the subset of memes used for the mutation and local search also prefers memes with low $P(M)$ and this way focuses on areas within a solution where improvements can be found more easily. After each iteration, the values for $B(M)$ and $P(M)$ are updated for each $M \in \mathcal{M}'$:

$$\begin{cases} B(M) = F(M), P(M) = 0, & \text{if } B(M) < F(M) \\ P(M) = P(M) + 1 & \text{otherwise} \end{cases}$$

3.4 Crossover and selection

First, an individual is created by selecting for each period the meme M with the best current fitness value $F(M)$ out of the current memepool. This individual is likely to become the elitist in the current population. However, shifts overlapping different memes might prevent this to happen. Using this procedure we make sure to keep the best memes.

Second, each of $|\mathcal{I}| - 1$ individuals is created as follows: For all sets of memes $\mathcal{M}$, each containing all memes with the same period $[m', m'')$ in the current memepool, we perform a k-tournament selection [5]: We select k memes out of $\mathcal{M}$. Out of these k memes, the meme with the lowest penalty value $P(M)$ is inherited to the offspring. The parent individuals are deleted and the algorithm continues using only the offsprings and their memes.

The first part assures to survive the best memes in the current memepool. The second part forms the actual crossover procedure. Using $P(M)$ as selection criteria, we get rid of memes that have been stuck in local optima for too long. If

a local optimum constitutes in fact a global optimum, then it survives through the first step of the crossover operator as described above.

Fig. 4 depicts the crossover operator: The first offspring is created by choosing only the fittest memes, i.e. M_1 from I_2 and M_2 from I_1. The remaining offsprings are created applying a k-tournament selection on memes with the same period. Different values for F after the crossover may occur from shifts overlapping into different memes.

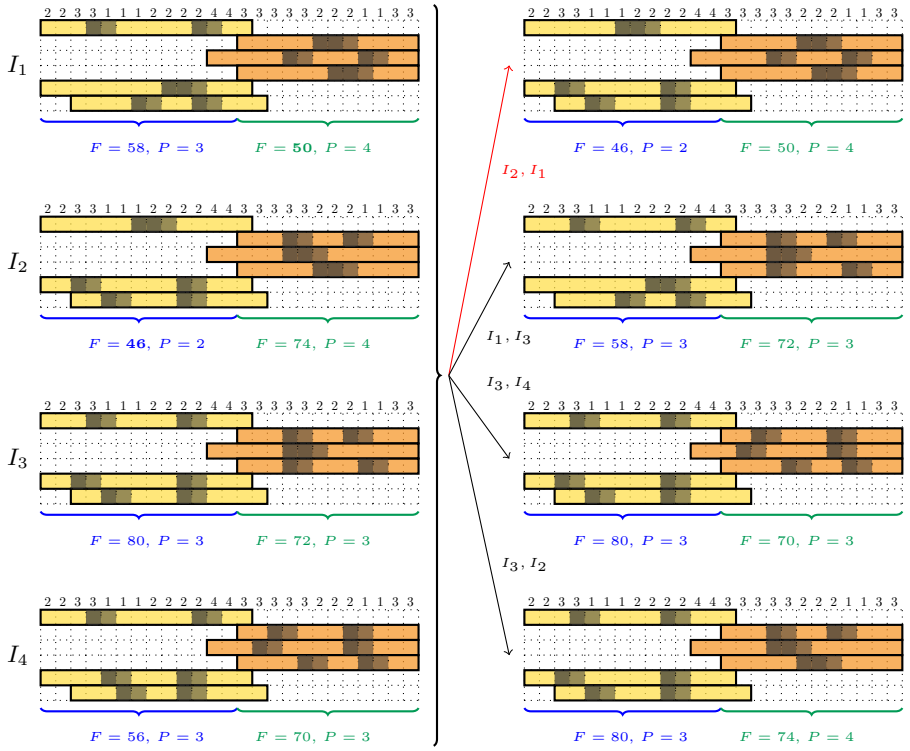

Fig. 4. Crossover operator

3.5 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 memes is defined s.t. $\mathcal{M}'$ contains the memes with the lowest penalty values (ties are broken randomly).

Each $M' \in \mathcal{M}'$ is mutated as follows: A set of shifts $\mathcal{S}' \in M'$ 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 $\mathcal{D}'_{S'}$ of break patterns computed in

the beginning (see Section 3.2). The size of $\mathcal{S}'$ is a parameter for which different values are evaluated in Section 4.

The local search is executed on each individual using set B of all breaks contained in $\mathcal{M}'$ for the respective individual. In 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 B. Second, a neighbourhood $\mathcal{N}$ out of three neighbourhoods $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ is chosen at random with a probability given by parameter $\eta = (\eta_1, \eta_2, \eta_3)$. Then the set $\mathcal{N}$ of all neighbours according to the chosen neighbourhood is computed. For each $N \in \mathcal{N}$ let $\delta(N, I) = F(N) - F(I)$, i.e. the difference between the fitness values. Let $\mathcal{N}' = \{N \in \mathcal{N} : \delta(N, I) \leq 0\}$. If $|\mathcal{N}'| > 0$ then $I = N$ with N being the best neighbour, i.e. neighbour $N \in \mathcal{N}'$ with minimal $\delta(N, I)$, ties broken randomly. Otherwise nothing happens. The local search terminates when for μ 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 μ and the probabilities η of the different neighbourhoods. Different values for these parameters and their evaluation are described in Section 4. Algorithm 2 outlines the local search procedure.

Algorithm 2 Local Search (Individual I, Breaks$\{\}B$)

```
 1: c ← 0
 2: repeat
 3:     b ← select break b ∈ B randomly
 4:     N ← select and compute one of {N_1, N_2, N_3} as moves from b
 5:     N' ← {N ∈ N : δ(N, I) ≤ 0}
 6:     if |N'| > 0 then
 7:         I ← N ∈ N' with minimal δ(N, I)
 8:         c ← 0
 9:     else
10:         c ← c + 1
11:     end if
12: until c == μ
13: return I
```

Neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$ are generated by the following three moves respectively: Single Assignment, Double Assignment, and Shift Assignment. The single assignment move assigns a break b to a different set of timeslots under consideration of $\mathcal{C}$. This includes appending b to its predecessor or successor, b' or b'' respectively, resulting in one longer break. The double assignment move involves two breaks. We consider a break b and both its predecessor b' and successor b'', or only b' respectively b'' if b is the last or first break within its shift. A double assignment move is a re-assignment of b and b' or b and b'' under consideration of $\mathcal{C}$. Like single assignment moves, two breaks might be joined to form a longer break. Two breaks of different length may also be swapped. A shift assignment

move assigns a whole break pattern $D \in \mathcal{D}'_S$ to the shift b is associated to. A detailed description of all moves and neighborhoods is given in [18].

4 Evaluation

We evaluated a set of parameters for which a preliminary analysis showed an impact on the solution qualities. Due to the execution time of the algorithm, it was not possible to do an exhaustive analysis. We thus started with a reasonable initial parameter setting and consecutively decreased and increased the value of each parameter while the other parameter values remained the same. The impact of each parameter was assessed using the Kruskal-Wallis test [11]. Details on the evaluations and values for each parameter setting are available in [18].

The parameter tests were conducted on six different instances among the 30 instances presented by Beer at al. [4], which are publicly available in [16]. 20 of the 30 instances in [4] were retrieved from a real life application without known optimal solutions, and ten selected among 60 randomly generated instances with known optimal solutions. Details regarding the random generation are provided by the same authors [16].

The input data $\mathcal{C}$ (constraints) and k (number of timeslots) are the same for all random and real life instances with $k = 2016$ and $\mathcal{C}$ defined as follows:

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 measures are given in timeslots with one timeslot corresponding to five minutes. k thus represents an entire calendar week.

The timeout was normalised to 3046 seconds according to a benchmark of the machine of [4] and ours. This allows us a more reliable comparison of our results to those of [4]. We ran the algorithm ten times for each instance and parameter value. Each run was performed on one core with 2.33Ghz of a QuadCore Intel Xeon 5345 with three runs being executed simultaneously, i.e. three cores being fully loaded. The machine provides 48GB of memory.

All experiments and the final runs we compare with [4] have been conducted using the mentioned timeout and hardware resources.

4.1 Parameter evaluation for our new algorithm (MAPBS)

The evaluated parameter values for our algorithm are given below:

$|\mathcal{I}|$ Population size, values tested: 1, 4, 6, 10, 20, best $|\mathcal{I}| = 4$
λ Defines number of memes $|\mathcal{M}'|$ being mutated and improved for each individual: $\max(1, |\mathcal{M}| \cdot \lambda)$, $0 \leq \sigma \leq 1$, values tested: 0.05, 0.1, 0.2, 0.3, 0.5, best $\lambda = 0.05$

σ Mutation weight, number of shifts being mutated: $\max(1, |S'| \cdot \sigma)$, $0 \leq \sigma \leq 1$, values tested: 0.01, 0.05, 0.1, 0.3, 0.5, best $\sigma = 0.05$

κ Selection: Number of memes performing a tournament in the crossover operator, values tested: 1, 2, best $\kappa = 1$ with significantly different results only for one out of six instances

μ Search intensity: Number of iterations the local search continues without finding improvements, this value is multiplied by the number of breaks $|B|$ available to the local search, values tested: 10, 20, 30, 40, best $\mu = 20$

(η_1, η_2, η_3)**:** Probability for each neighbourhood to be selected in local search iterations, 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 $\eta = (.8, .2, 0)$

The algorithm performs best with a small population size. 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. In this case no selection and no crossover is performed and thus the algorithm is reduced to a local search with mutation acting as perturbation. The search intensity μ was set to 20, the same value used for the population-based approach. After 20 iterations the algorithm performs a mutation on the individual as described in Section 3.5. The mutation prevents the search from getting trapped in a local optimum. Since mutation may worsen a solution during the progress of the algorithm, the best obtained solution is kept in memory. The results showed clearly that there is indeed the need for a population based approach, as the results with runs applying local search only, i.e. with $|\mathcal{I}| = 1$, gave the worse results.

To verify the need for hybridisation of the genetic operators with a local search mechanism, we additionally conducted experiments leaving out the local search. The rest of the algorithm was executed as described in the previous sections. The parameter settings for this experiment correspond to the settings for the final runs. On all tested instances the solution qualities significantly worsened.

The mutation and search rate λ determining $|\mathcal{M}'|$ led to the best results when kept low. On many instances, $\lambda = 0.05$ leads to only one meme being mutated and searched. The mutation weight σ also worked well with a low value. Different values for κ did not have a significant impact on most instances tested.

The local search intensity μ was set relative to the number of breaks $|B|$ taking part in the search. MAPBS considers only a subset of all breaks, namely those contained in $\mathcal{M}'$, which, according to the low value for λ are only a small subset. All runs where $\mathcal{N}_3$ participated gave worse results than those where we used only $\mathcal{N}_2$ and $\mathcal{N}_1$. The best performing combination was $\eta_1 = 0.8$, $\eta_2 = 0.2$ and $\eta_3 = 0$.

4.2 Final results and comparison with literature

Based on our experiments we used the following settings for the final runs: $|\mathcal{I}| = 4$, $\lambda = 0.05$, $\sigma = 0.05$, $\mu = 20$, $\kappa = 1$ and $\eta = (0.8, 0.2, 0.0)$. We

note that MAPBS outperforms significantly the results obtained with our previous memetic algorithm [13] and therefore we compare only results of our new memetic algorithm with the best existing results in the literature. Table 1 compares the results of MAPBS to state of the art results from [4]. Columns "Best" and "Average" respectively show best, average values of the objective function presented in the problem definition based on ten runs. Solutions provided by [4] have shown to be very good in practice and are currently in use in real life applications. Based on Table 1 we can conclude that our algorithm (MAPBS) returns improved results on all random instances and 18 out of 20 real life instances compared to results from literature.

The improvements for some random instances are striking. It is not clear if this is due to the algorithm itself or due to the initialisation process: Authors in [4] initialised their algorithm for the random instances without taking into account the set of constraints $\mathcal{C}$. The constraints were resolved, or partly resolved, in course of their algorithm. Our algorithm uses the approach proposed by [8] to initialise the solutions, and this possibly had an impact on these very significant improvements.

5 Conclusions

In this paper we introduced a new memetic algorithm to solve the break scheduling problem Bsp. We proposed a new memetic representation of Bsp, genetic operators and a local search heuristic. The local search heuristic with three different neighborhoods is applied only on selected memes. Further, we introduced a method based on penalty values to avoid local optima for parts of solutions.

We conducted a set of experiments for different parameter settings. The impact of each parameter was assessed with statistical methods. The use of genetic operators combined with the local search heuristic returned better results than using only local search. Applying the local search only on small parts of each individual significantly improved the qualities of the solutions compared to applying the local search on entire individuals. Focusing on neighbourhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ returned better solutions than using only one neighbourhood. The largest neighbourhood, $\mathcal{N}_3$, performed worst. The use of a penalty system along with focusing the local search only on memes that are not likely to be stuck in local optima significantly improved the solution qualities.

The results of the algorithm with parameter values performing best according to our experiments were compared to the results in literature on 30 publicly available benchmarks. Our algorithm returned improved results for 28 out of 30 instances. To the best of our knowledge, our method gives currently best results for the Bsp.

Acknowledgments: 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).

References

1. T. Aykin. A comparative evaluation of modelling approaches to the labour shift scheduling problem. *European Journal of Operational Research*, 125:381–397, 2000.
2. S. Bechtold and L. Jacobs. Implicit modelling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990.
3. A. Beer, J. Gärtner, N. Musliu, W. Schafhauser, and W. Slany. Scheduling breaks in shift plans for call centers. In *The 7th International Conference on the Practice and Theory of Automated Timetabling*, Montral, Canada, 2008.
4. A. Beer, J. Gärtner, N. Musliu, W. Schafhauser, and W. Slany. An AI-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25(2):60–73, 2010.
5. A. Brindle. *Genetic algorithms for function optimisation.* PhD thesis, University of Alberta, Department of Computer Science, Edmonton, Canada, 1981.
6. C. Canon. Personnel scheduling in the call center industry. *4OR: A Quarterly Journal of Operations Research*, 5(5(1)):89–92, 1989.
7. M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 2009.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
9. L. Di Gaspero, J. Gärtner, N. Musliu, A. Schaerf, W. Schafhauser, and W. Slany. A hybrid LS-CP solver for the shifts and breaks design problem. In *The 7th International Workshop on Hybrid Metaheuristics (HM 2010). Lecture Notes in Computer Science. To appear*, Vienna, Austria, 2010.
10. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, 1989.
11. D. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, 6 edition, 2005.
12. P. Moscato. On evolution, search, optimization, gas and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Comput. Prog. Rep. 826, California Institute of Technology, 1989.
13. N. Musliu, W. Schafhauser, and M. Widl. A memetic algorithm for a break scheduling problem. In *8th Metaheuristic International Conference*, Hamburg, Germany, 2009.
14. C.-G. Quimper and L.-M. Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, 16(3):373–391, 2010.
15. M. Rekik, J. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13:49–75, 2010.
16. http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/, 2008.
17. P. Tellier and G. White. Generating personnel schedules in an industrial setting using a tabu search algorithm. In H. R. E. K. Burke, editor, *PATAT 2006*, pages 293–302, 2006.
18. M. Widl. Memetic algorithms for break scheduling. Master's thesis, Vienna University of Technology, Vienna, Austria, 2010.

Instance	Best existing results [4]			Results of MAPBS		
	Best	Average	σ	Best	Average	σ
2fc04a	3,094	3,224	84	**2,816**	**2,961**	71
2fc04a03	3,100	3,200	61	**2,834**	**2,934**	54
2fc04a04	3,232	3,342	68	**2,884**	**2,954**	60
2fc04b	**1,822**	2,043	92	1,884	**1,948**	49
3fc04a	1,644	1,767	102	**1,430**	**1,533**	67
3fc04a03	1,632	1,759	87	**1,440**	**1,514**	40
3fc04a04	1,932	1,980	40	**1,614**	**1,718**	48
3si2ji2	3,626	3,667	35	**3,177**	**3,206**	17
4fc04a	1,694	1,817	126	**1,478**	**1,540**	29
4fc04a03	1,666	1,795	87	**1,430**	**1,502**	42
4fc04a04	1,918	2,017	95	**1,606**	**1,674**	48
4fc04b	1,410	1,489	49	**1,162**	**1,233**	48
50fc04a	1,672	1,827	81	**1,548**	**1,603**	36
50fc04a03	1,686	1,813	84	**1,402**	**1,514**	67
50fc04a04	1,790	1,917	64	**1,480**	**1,623**	89
50fc04b	1,822	2,012	91	**1,818**	**1,900**	56
51fc04a	2,048	2,166	89	**1,886**	**2,074**	87
51fc04a03	1,950	2,050	86	**1,886**	**1,949**	46
51fc04a04	2,058	2,191	64	**1,958**	**2,039**	52
51fc04b	**2,244**	2,389	94	2,306	**2,367**	43
random1-1	728	972	177	**346**	**440**	48
random1-2	1,654	1,994	172	**370**	**476**	65
random1-5	1,284	1,477	99	**378**	**418**	29
random1-7	860	1,077	154	**496**	**583**	42
random1-9	1,358	1,658	213	**318**	**423**	51
random1-13	1,264	1,535	245	**370**	**445**	55
random1-24	1,586	1,713	74	**542**	**611**	43
random1-28	1,710	2,020	233	**222**	**318**	71
random2-1	1,686	1,855	142	**724**	**889**	75
random2-4	1,712	2,053	242	**476**	**535**	45

Table 1. Comparison with literature: Real life and random instances