A Memetic Algorithm for a Break Scheduling Problem

Nysret Musliu Werner Schafhauser Magdalena Widl

Institute of Information Systems, Vienna University of Technology
Favoritenstrasse 9, 1040 Wien, Austria
{musliu, schafha, widl}@dbai.tuwien.ac.at

Abstract

In this paper we regard a break scheduling problem originating in the area of supervision personnel. This complex task requires the assignment of several breaks per shift, satisfying various constraints reflecting labour rules, staffing requirements and ergonomic criteria. To solve this problem we propose a memetic algorithm combining a genetic algorithm and a local search procedure. We evaluate the presented memetic algorithm on publicly available benchmark instances and compare the obtained results to an existing min-conflicts based algorithm. The memetic algorithm is able to return competitive results and for half of the regarded instances it computes solutions of improved quality.

1 Introduction

Break scheduling problems emerge in many working areas where breaks are indispensable due to characteristics of the tasks to be performed. These characteristics include the requirement of high concentration during long periods of time, continuous work in front of computer monitors or other monotonic and exhaustive activities. Typically, break scheduling problems arise in call centres, security checking or assembly lines.

In this article we regard a real-life break scheduling problem where various breaks have to be inserted into an already existing shift plan for supervision personnel [**?**]. The breaks should be scheduled in a way such that several constraints reflecting staffing requirements, labour rules and ergonomic criteria are satisfied completely or violated only to a minimum degree. Obtaining solutions of satisfactory quality is crucial for the acceptance of the resulting shift plans by the affected employees and minimises the costs for human resources.

In literature, break scheduling problems have been hardly addressed on their own, but they can be found along with the so-called shift scheduling problem. Dantzig developed the original set-covering formulation [5], in which feasible shifts are enumerated based on possible shift starts, shift durations, breaks, and time windows for breaks. Bechtold and Jacobs proposed a new integer programming formulation [2] and model break placements implicitly. Thompson introduced a fully implicit formulation of the labour shift scheduling problem [11]. Aykin compares different shift scheduling models with breaks in [1]. Besides exact methods, also meta-heuristic methods, such as

tabu search have been proposed for scheduling problems similar to the one addressed in this work and returned satisfactory results [4, 10]. The break scheduling problem for supervision personnel has already been solved with a break scheduling system applying a min-conflicts based local search algorithm [**?**]. This algorithm iteratively improves the current solution by concentrating only on breaks causing constraint violations. During each iteration this heuristic selects randomly a break violating a constraint and constructs its neighborhood. The soultion minimizing or at least not worsening the violation degree of the current solution is selected for the next iteration. Additionally, to avoid a local optimum the authors applied in each iteration with some probability the random walk strategy. The random walk strategy applies a random move to a break that is selected from the set of all breaks that violate constraints.

In this work we propose a memetic algorithm to obtain solutions of improved quality for the break scheduling problem for supervision personnel. Our algorithm prevents the violation of most constraints and assures that a returned solution conforms with labour rules and ergonomic criteria. For our memetic algorithm we propose two crossover operators, a mutation operator, and a selection mechanism to obtain solutions for the next generation. We enhance the basic genetic algorithm by a local search procedure to improve a determined number of solutions in each iteration. The local search procedure is based on a min-conflicts heuristic with three different neighbourhoods. Additionally, we propose to enumerate possible break patterns for shifts with equal start and duration satisfying a number of constraints. These patterns are used for the initialisation process and in one of the neighbourhoods. Our hybrid algorithm is experimentally evaluated for benchmark examples from literature. A comparison with existing results shows that the memetic algorithm is able to improve the existing results for 50% of all benchmark instances.

2 Problem Definition

In the break scheduling problem for supervision personnel [**?**] we regard shift plans for supervision personnel in which each shift must contain a certain amount of break time. Our goal is to schedule breaks within the shifts in such a way that we obtain a solution minimising a weighted sum of constraint violations. These constraints represent legal demands, staffing requirements and ergonomic criteria and the desired solution should violate them only to a minimum degree. Formally, as input for the break scheduling problem we are given:

- a planning period formed by T consecutive time slots $[a_1, a_2), [a_2, a_3), ..., [a_T, a_{T+1}]$ all having the same slot length of five minutes. The time points a_1 and a_{T+1} represent the beginning and end of the planning period.
- n shifts $s_1, s_2, ..., s_n$ representing employees working within the planning period. Each shift s_i has the following adjoined parameters: $s_i.breaktime$ specifying the required amount of break time for s_i in time slots, $s_i.start$ and $s_i.end$, representing the time slots in which a shift starts and ends in. The value of a further parameter $s_i.duration$ can be derived by subtracting the value of $s_i.start$ from $s_i.end$.
- the staffing requirements for the planning period. For each time slot $[a_t, a_{t+1})$ we are given an integer value r_t indicating the optimal number of employees that should be working during time slot $[a_t, a_{t+1})$. An employee is considered to be working during time slot $[a_t, a_{t+1})$ if

neither he/she has a break during time slot $[a_t, a_{t+1})$ nor his/her break has ended at time point a_t. After a break an employee needs a full time slot, usually five minutes, to reacquaint him- or herself with the altered situation. Thus also during the first time slot following a break an employee is not considered to be working.

Each break b_j is associated to a certain shift and has three adjoined parameters $b_j.start$, $b_j.end$, and $b_j.duration$. We distinguish between two different types of breaks: lunch breaks and monitor breaks. Given a planning period, a set of shifts, the associated total break times and the staffing requirements, a *feasible solution* to the break scheduling problem is a set of breaks such that:

- each break b_j lies entirely within its associated shift s_i.
- two distinct breaks b_j, b_k associated with the same shift s_i do not overlap in time.
- in each shift s_i the sum of durations of its associated breaks equals the required amount of break time. $\sum_{b_j \in s_i} b_j.duration = s_i.breaktime$

Among all feasible solutions for the break scheduling problem we aim at finding an optimal one according to various criteria. These criteria are modelled as soft constraints on a solution and may be described as follows:

C_1 **Break Positions:** Each break b_j may start at the earliest a certain number of time slots after the beginning of its associated shift s_i and it may end at the latest a given number of time slots before the end of its shift.

C_2 **Lunch Breaks:** A shift s_i can have several lunch breaks, each of which is required to last a specified number of time slots and should be located within a certain time window after the shift start.

C_3 **Duration of Work Periods:** Breaks divide a shift into several work and rest periods. The duration of work periods within a shift must range between a required minimum and a maximum duration.

C_4 **Minimum Break Times:** If the duration of a work period exceeds a certain limit, the break following that period must last a given minimum number of time slots (*min. ts count*).

C_5 **Break Durations:** The duration of each break b_j must lie within a specified minimum and maximum value.

C_6 **Shortage of Employees:** In each time slot $[a_t, a_{t+1})$ at least r_t employees should be working.

C_7 **Excess of Employees:** In each time slot $[a_t, a_{t+1})$ at most r_t employees should be working.

Objective function For each constraint we define a violation degree $violation(C_k)$ specifying the deviation, in time slots or employees, from the requirements stated by the respective constraint. The importance of a single criterion and the corresponding constraint varies from task to task. Consequently, the break scheduling problems objective function is the weighted sum of the violation degree of each constraint, or more formally:

$$F(Solution) = \sum_{k=1}^{7} W_k \times violation(C_k)$$

3 Solving the Break Scheduling Problem with a Memetic Algorithm

The term Memetic Algorithms was first used by Moscato in [7]. Memetic algorithms combine population bases strategies (genetic algorithms and other evolutionary algorithms) with local search to solve optimisation problems. Genetic algorithms imitate natural evolution on search and optimisation problems. Improvements on a pool of individuals, each representing a sub-optimal solution, are to be found by applying genetic operators such as selection, mutation and crossover. The initial population is usually created randomly or using some kind of heuristic [8].

In this section we propose a memetic algorithm that combines a genetic algorithm with a local search mechanism on a subset of individuals to solve the break scheduling problem. The algorithm is outlined in Algorithm 1.

Algorithm 1 Memetic Algorithm

```
1: I ← initialise population with n individuals
2: EVALUATE(each individual i ∈ I)
3: repeat
4:     e ← fittest individual i ∈ I
5:     I ← TOURNAMENTSELECT(I) ∪ e
6:     for all individuals i ∈ I \ e do
7:         x ← select random float uniformly distributed in [0..1]
8:         if x ≤ α then
9:             j ← select random individual j ≠ i
10:            i ← CROSSOVER(i, j)
11:        else
12:            i ← MUTATE(i)
13:        end if
14:    end for
15:    EVALUATE(each individual i ∈ I)
16:    m ← |I| · λ
17:    L ← m fittest individuals in I
18:    for all l ∈ L do
19:        l ← LOCALSEARCH(l)
20:    end for
21: until timeout
22: return best solution in generation
```

Fitness function The fitness function determining the fitness of each individual corresponds to the objective function described in Section 2.

Solution representation A solution is represented by a set of breaks for each shift in the shift plan. A break consists of two integers representing break start and duration.

Shift domains For each shift s there is a number of possible break patterns satisfying constraints C_1-C_5. We refer to the set of all break patterns satisfying constraints C_1-C_5 as the shift's domain. Shifts with equal values for *s.duration* and *s.breaktime* share the same domain. The problem of finding a break pattern within the shift domain can be modelled as small temporal constraint satisfaction problem and consequently can be solved in polynomial time by applying Floyd-Warshall's shortest path algorithm [9, 6]. We use this fact to precalculate a subset of each shift domain and will refer to it by $D(s)$. $D(s)$ contains all break patterns that can be formed for shift s by using only breaks of minimal duration (as specified in constraint C_5). Figure 1 shows an example of three different break patterns.

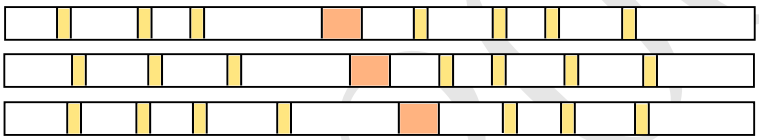

Figure 1: Possible break patterns satisfying C_1-C_5

The size of $D(s)$ strongly depends on the relation between *s.duration* and *s.breaktime*. Due to the restrictions imposed by constraints C_1-C_5, a long shift with a proportionally short breaktime results in a small number of possible break patterns. Overall, the sizes of the domain subsets for the sample instances we regard in this paper range between 65 and 115.000. We use the precalculated domain subsets in the initialisation and local improvement process, as described in later paragraphs.

Initialisation Each individual in the population is initialised in two steps: Firstly, for each shift s a valid break pattern of breaks $d \in D(s)$ is selected randomly. This provides us with a first solution that fulfills constraints C_1-C_5. Secondly, a randomised local search procedure is executed on the solution.

The randomised local search randomly picks a break and performs a random improving step in the break's neighbourhood. For this process we use the *single assignment* neighbourhood as described in the local search paragraph.

The first initialisation step assures a high diversity in the gene pool, while the latter is used to establish starting solutions of fairly good fitness. By using a randomised local search, we avoid decreasing the diversity.

Note that constraints C_1-C_5 remain satisfied in all phases of our algorithm. Therefore, a particular move will be applied in the solution only if it does not violate these constraints.

Selection The selection operator decides which individuals of the current generation are allowed to enter the next generation. Firstly, an elitist individual is determined by selecting the individual with the best fitness value in the current generation (ties are broken randomly). The elitist individual will remain unaltered during the whole iteration.

Secondly, individuals are selected out of the current generation by a k-tournament selector [3]: k individuals are selected randomly to perform a tournament. The individual with the best fitness value wins and will serve as a crossover or mutation candidate in the next step. This procedure is repeated $n-1$ times, n being the population size ($n-1$ since one individual is represented by the elitist). The generation now consists of the elitist and the individuals selected by the tournament selection operator.

Mutation and Crossover: In the next step of the algorithm, the crossover or the mutation operator is applied on each individual i in the generation, except for the elitist individual. Crossover will take place with probability α and mutation with $1-\alpha$. We propose a combination of two different crossover operators. Both select a partner different to i randomly out of the generation and create an offspring inheriting entire shifts with their breaks from either of the parents. They differ in the decision, which shift to take from which partner.

Simple Crossover: The offspring is produced by randomly inheriting a percentage γ of all shifts from one parent's exchangeable shifts, and the remaining shifts from the other parent. By exchangeable shifts we refer to shifts with equal shift start, shift duration and breaktime.

Smart Crossover: From the parent whose fitness value is worse, we determine a set of shifts highly involved in constraint violations. That is, a set of shifts covering periods which include timeslots with high shortage of working employees. The offspring inherits this set of shifts from the better parent's exchangeable shifts and the remaining shifts from the worse.

Mutation: The mutation operator performs one random move on the given individual using the *single assignment* neighbourhood, which is described in the local search paragraph.

Local Search: In each iteration, the m individuals, $m = populationSize \cdot \lambda$ of the current generation are locally improved by the following local search procedure: In each iteration, a break b is selected randomly out of the solution. Then a neighbourhood is selected randomly out of the three different neighbourhoods proposed below. In this neighbourhood the most improving move is performed. If no improving move can be found, we continue by selecting another break b at random. The local search terminates if for a given number of subsequent iterations no improving move can be found.

Neighbourhoods:

Single Assignment: This neighbourhood comprises all moves for a break b, in which b is assigned a new value for $b.start$ without violating any of the constraints C_1-C_5. This move may also append b to its successor, creating a longer break period and consequently causing a longer work period, as long as this does not violate any of C_1-C_5 (see Figure 2).

Figure 2: Single assignment

Double Assignment: In the second neighbourhood, two breaks are involved. It consists of all possible moves of b and its predecessing break b' in the same shift. Similarly to the first neighbourhood, two breaks may be connected to form a longer break period. In case the two breaks are of different duration, it also may happen the breaks being swapped (see Figure 3 and Figure 4).

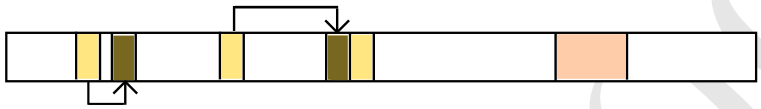

Figure 3: Double assignment

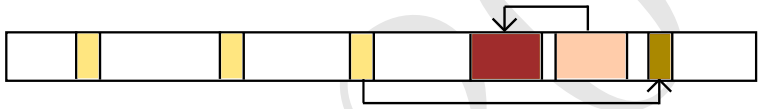

Figure 4: Double assignment with swap

Shift Assignment: The third neighbourhood makes use of the precalculated domain subsets described above. It determines the shift s to which b is associated and selects randomly a subset $D' \subseteq D(s)$. For performance reasons, $|D'|$ may not exceed a certain limit. One move in this neighbourhood is performed reassigning all breaks in s according to $d \in D'$.

We have tested the algorithm using only one neighbourhood, combinations of two and a combination of three neighbourhoods. In the cases of combination, on each iteration a neighbourhood is being selected randomly. A combination of all neighbourhoods led to the best results as it leads the search to different areas in the search space and thus prevents being trapped early in local optima.

4 Experiments

Experiments have been conducted on randomly created instances which can be found in `http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/verb`. Explications on the generation of random instances are available on this website, too.

Each experiment was executed on one core of a QuadCore Intel Xeon 5345 and with 48GB RAM. We benchmarked this machine in order to compare with results in [**?**]. According to this benchmark, the runtime of our algorithm was 18 minutes (for each run of the algorithm).

Parameter Settings Various parameters strongly influence the performance of the algorithm, hence several experiments to determine the final parameter settings have been executed.

Population size: Experiments have been conducted with population sizes of 10, 20, 30, 40 and 70. The size of 40 returned the best results. 40 individuals are enough to keep a sufficiently high diversity, while a higher population size slows down the algorithm significantly.

Crossover/mutation rates: The probability an individual is created by the crossover operator has been tested with $\alpha = 0.7$ and $\alpha = 0.9$ where $\alpha = 0.9$ clearly was more competitive. The reason for this behaviour probably lies in mutation only performing very small steps out of neighbourhoods that are used in the local search in any case, while the crossover operator constructs different solutions which provide good and diverse starting points for the local search.

Tournament selection 'k': The tournament selector has been tested with $k = 2$ and $k = 3$, the former outperforming the latter on all instances. Again, it appears like the reason for this lies in diversity of the solution pool. $k = 3$ results in lower diversity and faster stagnation.

Local search rate: The local search rate determines the number of individuals to be locally improved relative to the population size. Experimental settings have been $\lambda = 0.1$, $\lambda = 0.15$ and $\lambda = 0.2$. $\lambda = 0.2$ caused the algorithm to stagnate in an early stage, while $\lambda = 0.1$ and $\lambda = 0.15$ performed well, differing slightly depending on the instance. For the experiments conducted for this paper we chose $\lambda = 0.15$.

Local search intensity: By local search intensity τ we refer to the number of iterations without improvements causing the local search to terminate. We tested with $\tau = 100$, $\tau = 500$, $\tau = 700$ and $\tau = 1000$. $\tau = 100$ turned out too low, returning only small improvements on each iteration and thus requiring more iterations for a result that could be found quicker using $\tau = 500$ or $\tau = 700$. Raising the value for τ trades off higher improvement per iteration for a longer search time. We found the best balance in $\tau = 700$.

Crossover types Two different crossover types are described in Section 3. We have performed experiments using only simple or smart crossover, as well as combinations of both. The best results have been achieved combining both types with $\gamma = 0.9$ for smart crossover to be selected.

Results and comparison with literature The results for the final series of experiments are described in Table 1. We report the best and mean fitness obtained in ten runs for each instance along with the respective standard deviations. Additionally, Table 1 shows the results returned by the min-conflicts-based local search algorithm in [**?**] and marks the best fitness values per instance in bold style. For half of the regarded benchmark instances the memetic algorithm outperformed the algorithm described in [**?**].

5 Conclusion

We have introduced a memetic algorithm to obtain solutions of satisfactory quality for a complex break-scheduling problem originating in the area of supervision personnel. This algorithm consists of the three standard operators selection, crossover and mutation and is hybridised with a min-conflicts search. Initial solutions are constructed with break patterns already fulfilling constraints

Instance	Shifts	Breaks	Hybrid GA			Min-Conflicts-Random-Walk [?]		
			Best	Average	Std.Dev	Best	Average	Std.Dev.
random1-1	137	962	**672**	**738**	56	728	972	177
random1-2	164	1060	2174	2352	89	**1654**	**1994**	172
random1-5	141	950	**870**	**1044**	86	1284	1477	99
random1-7	157	1089	**742**	**861**	68	860	1077	154
random1-9	151	985	1918	2095	108	**1358**	**1658**	213
random1-13	124	884	2884	3039	98	**1264**	**1535**	245
random1-24	137	928	**1182**	**1330**	107	1586	1712	75
random1-28	124	809	2926	3018	63	**1710**	**2020**	233
random2-1	179	1255	**1262**	**1537**	117	1686	1855	142
random2-4	162	1075	2182	2336	111	**1712**	**2053**	242

Table 1: Comparison with literature

C_1-C_5, representing labour rules and ergonomic criteria. These constraints remain unviolated during the entire run of the algorithm.

In each iteration, the genetic operators generate a pool of different solutions out of the previous generation. The best solutions are further optimised by the local search procedure. This way, the entire generation moves towards better solutions without significantly reducing the diversity. High diversity produced by the genetic operators allows the local search to exploit different areas of the search space and thus avoids early stagnation.

Based on our experimental results we can conclude that for the break shceduling problem the results of genetic algorithms can be significantly improved if they are used in combination with local search. Using of break patterns for inititial solutions and for generation of neighborhood has shown to improve further these results.

For the future work we will consider improving of local search procedure by including a tabu mechanism and we plan to analyze more deeply the impact of each neighborhood used in the local search procedure.

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

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.E. Bechtold and L.W. Jacobs. Implicit modelling of flexible break assignments in optimal shift

scheduling Management Science, 36(11):1339-1351, 1990.

[3] A. Brindle. Genetic algorithms for function optimisation PhD thesis, University of Alberta, Department of Computer Science, Edmonton, Canada.

[4] C. Canon. Personnel scheduling in the Call Center industry 4OR: A Quarterly Journal of Operations Research, 5(1):89-92, 1989.

[5] G. B. Dantzig. A comment on Eddie's traffic delays at toll booths Operations Research, 2:339-341, 1954.

[6] R. Dechter, I.Meiri, J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61-95, 1991.

[7] P. Moscato. On evolution, search, optimization, GAs and martial arts: toward memetic algorithms Pasadena Tech. Rep. Caltech Concurrent Comput. Prog. Rep. 826, California Institute of Technology, 1989.

[8] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning Addison-Wesley Publishing Company, 1989.

[9] C. H. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity Prentice Hall, 1982.

[10] P. Tellier and G. White. Generating Personnel Schedules in an Industrial Setting Using a Tabu Search Algorithm E. K. Burke, H. Rudov (Eds.): PATAT 2006, 293-302, 2006.

[11] G. Thompson. Improved implicit modeling of the labor shift scheduling problem Management Science, 41(4):595-607, 1995.