FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Memetic Algorithms for Break Scheduling

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Magdalena Widl
Matrikelnummer 0327059

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Priv.-Doz. Dr. Nysret Musliu
Mitwirkung: Dipl.-Ing. Werner Schafhauser

Wien, 10.04.2010

(Unterschrift Verfasserin)	(Unterschrift Betreuer)

DECLARATION

Magdalena Widl
Parkweg 24
2352 Gumpoldskirchen

"Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 1. April 2010

Magdalena Widl

Meiner Oma gewidmet

ABSTRACT

Break scheduling problems arise in working areas where breaks are indispensable due to the nature of the tasks to be performed, e.g. in air traffic control, supervision or assembly lines. We regard such a problem, denoted Bsp, originating in the area of supervision personnel. The objective is to assign breaks to an existing shiftplan such that various constraints reflecting legal demands or ergonomic criteria are satisfied and staffing requirement violations are minimised.

We prove NP-completeness of this problem when all possible break patterns are given explicitly in the input.

To solve Bsp, we propose variations of a memetic algorithm. A memetic algorithm combines genetic operators such as selection, crossover and mutation with local improvement techniques. We suggest two different memetic representations of Bsp and three different memetic algorithms built upon these representations. The algorithms differ in their genetic operators and memetic representation, but include the same local search heuristic. We propose three different neighbourhoods the local search can be based upon. Additionally, we evaluate the impact of a tabu list within one of the algorithms, and within another the impact of a penalty system that tries to detect local optima.

Our approaches are influenced by various parameters, for which we experimentally evaluate different settings. The impact of each parameter is assessed with statistical methods. We compare the three algorithms, each with the best parameter setting according to the evaluation, to state of the art results on a set of publicly available instances. 20 instances were drawn from real life scenarios and 10 randomly generated. 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.

ZUSAMMENFASSUNG

Pauseneinteilungsprobleme entstehen in Arbeitsbereichen, in welchen regelmäßige Pausen unabdingbar sind. Dazu zählen Flugsicherung, Überwachungsaufgaben oder Fließbandarbeit. Wir betrachten ein solches Problem, abgekürzt Bsp für die englische Bezeichnung "Break Scheduling Problem", aus dem Überwachungsbereich. Das Ziel ist es, Pausen in einen gegebenen Schichtplan unter Berücksichtigung verschiedener rechtlicher und ergonomischer Kriterien so einzuteilen, dass Abweichungen vom gegebenen Personalbedarf minimiert werden.

Wir zeigen, dass Bsp NP-vollständig ist, wenn eine fixe Menge an Pausenmustern Teil des Inputs ist.

Zur Optimierung des Bsp entwickeln wir verschiedene memetische Algorithmen. Ein memetischer Algorithmus kombiniert genetische Operatoren wie Selektion, Kreuzung und Mutation mit lokalen Verbesserungstechniken. Wir präsentieren zwei verschiedene memetische Darstellungen des Bsp sowie, darauf aufbauend, drei verschiedene neuartige memetische Algorithmen. Die Algorithmen unterscheiden sich durch ihre genetischen Operatoren und die memetische Darstellung, verwenden aber dieselbe lokale Suche als Verbesserungstechnik. Für die lokale Suche präsentieren wir drei verschiedene Nachbarschaften, die unterschiedlich kombiniert werden können. Wir evaluieren für einen der Algorithmen den Einfluss einer in den Suchprozess integrierten Tabu Liste, sowie für einen anderen Algorithmus den Einfluss der Vergabe von Strafpunkten zur Vermeidung lokaler Optima.

Der Verlauf dieser metaheuristischen Ansätze wird von mehreren Parametern beeinflusst, für welche wir verschiedene Werte experimentell evaluieren. Die Bedeutung jedes Parameters wird mithilfe statistischer Methoden beurteilt.

Wir vergleichen unsere Algorithmen, jeweils unter der besten ermittelten Parameterbelegung, mit Resultaten aus der Literatur auf öffentlich verfügbaren Instanzen. 20 Instanzen stammen aus Szenarien der Praxis und 10 sind zufällig generiert. Unser bester Algorithmus findet bessere Lösungen für 28 von 30 Instanzen. Nach unserem Wissen stellen unsere verbesserten Resultate für die Instanzen aus der Praxis neue obere Schranken dar.

PUBLICATIONS

Some results of this thesis have appeared previously in the following publication:

Nysret Musliu, Werner Schafhauser and Magdalena Widl: *A Memetic Algorithm for a Break Scheduling Problem*. The 8th Metaheuristic International Conference (MIC 2009), Hamburg, Germany, July 13-16, 2009.

ACKNOWLEDGMENTS

I hereby express my gratitude and sincere appreciation to my advisor Nysret Musliu, whose expertise, support, encouragement and patience were invaluable for the progress of this work. I thank my co-advisor Werner Schafhauser for his assistance in many areas, be it writing of a publication, algorithmic questions or implementation issues. I also really enjoyed his particular kind of humour.

I thank the DBAI group of the Institute of Information Systems for providing me with a workstation, a quiet room, powerful servers and a coffee machine. A special reference goes to the group's system administrator Toni Pisjak for his technical support and never-failing backup system.

During more than one year working within this group, I enjoyed the company and intellectual input of many colleagues who in one way or another have influenced this work. In this context I particularly thank Andreas Pfandler, Emanuel Sallinger and Stefan Rümmele for their advice in complexity theory.

I am grateful to my parents and grandmothers for providing me with good education, encouragement and security during the past 26 years. I particularly thank my father for his interest and assistance regarding various mathematical questions that emerged during my studies.

Last but not least, *muchísimas gracias* go to my partner José for sharing with me both moments of joy and moments of difficulties.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

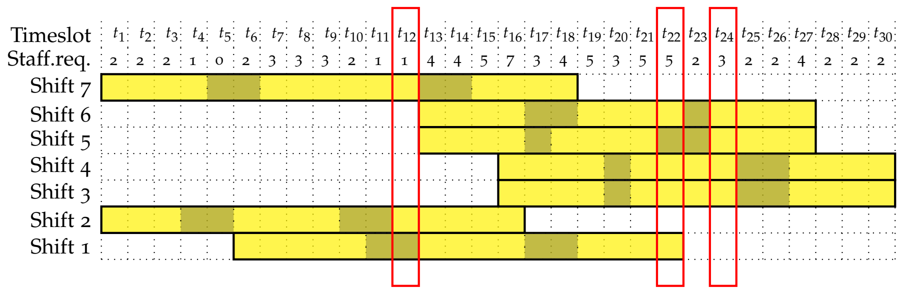
1 INTRODUCTION

Many working areas require staff to maintain high concentration when performing their tasks. This includes air traffic control, security checking, supervision or assembly line workers. Loss of concentration might result in dangerous situations. It is thus required, often by law, that staff working in such areas take breaks after given time periods.

Breaks are periods during working shifts when 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, i.e. for Austria, this can be found in paragraph 11 of Arbeitszeitgesetz [25]. 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 any 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 required number of employees is monitoring the screens. In order to avoid overestaffing and thus minimise personnel costs, we may also aim at an exact number of employees being present, instead of a minimum.

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


Timeslot
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16} t_{17} t_{18} t_{19} t_{20} t_{21} t_{22} t_{23} t_{24} t_{25} t_{26} t_{27} t_{28} t_{29} t_{30}
Staff.req.
2 2 2 1 0 2 3 3 3 2 1 1 4 4 5 7 3 4 5 3 5 5 2 3 2 2 4 2 2 2
Shift 7
Shift 6
Shift 5
Shift 4
Shift 3
Shift 2
Shift 1

Breaks are scheduled s.t. no break is longer than two timeslots, and no working period is longer than six timeslots. The staffing requirements are not satisfied in all timeslots, i.e. there is undercover in timeslot t_{22} and overcover in timeslots t_{12} and t_{24}.

Figure 1.1: Shiftplan with different shifts and breaktimes

maximum values limiting the length of breaks and worktime, to which we 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. A shiftplan with a possible, though sub-optimal, solution is depicted in figure 1.1. We denote our formulation as BREAK SCHEDULING PROBLEM and abbreviate it with BSP.

In literature, this problem has mainly been addressed as part of the so-called shift scheduling problem, where shifts are scheduled along with breaks. For this problem, there has been some previous work, like a set-covering formulation developed by Dantzig, [12], an integer programming formulation with implicit modelling of breaks by Bechtold and Jacobs [4], or heuristic methods suggested by Canon [11] and Tellier and White [32]. However, the problem of shift scheduling with breaks, as defined in the mentioned works, differs significantly to our case. BSP considers scheduled shifts are part of the input and the objective is to insert breaks according to staffing requirements and temporal constraints.

For BSP there has been previous work by Beer et al. [5, 6], who suggest a min-conflicts based metaheuristic and present experimental results on real life and randomly created instances. This previous work however leaves some open questions: What is the computational complexity of BSP? How do other methods perform on BSP? And is there a method that is able to improve the current results on BSP?

1.1 OBJECTIVES

The objectives of this work are:

- Determination of computational complexity for Bsp.
- Design and implementation of a metaheuristic based on the concept of memetics to obtain good solutions for Bsp. This includes the definition of a memetic representation, the crossover, mutation and selection operators as well as a method for local improvements.
- Determination and experimental evaluation of parameters that influence the process of the designed method.
- Comparison to state of the art results.

1.2 RESULTS

The following are the main contributions of this thesis:

- We prove that Bsp is NP-complete when break patterns are defined explicitely as part of the input.
- We propose three algorithms to optimise the break scheduling problem. Each of them is based on the concept of memetic algorithms introduced by Moscato [23]. The key idea is to improve a set of solutions, which have been created randomly or by some quick heuristic, by applying genetic operators together with some local improvement technique. We propose two different memetic representations, a set of genetic operators, a penalty system to detect local optima, as well as a local search heuristic based on different neighbourhoods and an optional tabu list.
- We experimentally evaluate the parameters that influence the optimisation process. The impact of each parameter is statistically verified.
- We compare our algorithms with the best existing results for this problem in literature. One of our algorithms returns improved results on 28 out of 30 instances. To our best knowledge, these results represent new upper bounds.

1.3 ORGANISATION

The remaining parts of this thesis are organised as follows: We first give a formal problem definition of BSP , present our complexity results and an give an overview on related work in Chapter 2. The concept of memetics and memetic algorithms is explained in Chapter 3. Chapter 4 provides an in-depth description of the memetic representations, the local improvement technique and the three proposed algorithms. Details on the experimental parameter optimisation, the nature of the real life and randomly created instances and a comparison with literature is given in Chapter 5. We draw our conclusions in Chapter 6.

2 THE BREAK SCHEDULING PROBLEM

2.1 PROBLEM STATEMENT

The break scheduling problem (Bsp) regards a shiftplan that consists of consecutive *timeslots* and of *shifts* starting and ending in defined timeslots (the length is the difference between start and end). One shift represents exactly one employee on duty. 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*. A slot can be assigned one of three values: break, worktime or time used for familiarisation with a new working situation. The objective is to find an assignment for each slot, such that breaks are distributed within each shift according to given criteria, while violations of staffing requirements, which are given for each timeslot, are minimised.

We first define relevant terms for Bsp and then continue with a formal problem definition.

Definition 2.1 (Timeslot)**.** A *timeslot* is a time period of fixed length. In our real life problem, one timeslot corresponds to a period of five minutes.

Definition 2.2 (Shift)**.** A *shift* S is defined by a set of n consecutive timeslots $S = \{t_i, t_{i+1}, ..., t_{i+n}\}$: $\forall j(i \leq j < i+n)$ it holds that $t_{j+1} - t_j = 1$. $S_s = t_i$ denotes the shift start and $S_e = t_{i+n}$ denotes the shift end. Each shift represents exactly one employee on duty.

Definition 2.3 (Slot)**.** A *slot* is a timeslot in a particular shift. A slot can be assigned one of three values: 1 (1-*slot*) for a working employee, 0 (0-*slot*) for an employee on break or $0'$ ($0'$-*slot*). $0'$-slots are assigned to those and only those slots that directly follow a sequence of 0-slots. A $0'$-slot stands for an employee who is getting familiar with an altered working situation after a break. During a $0'$-slot, the employee is not consuming breaktime but neither counted as working regarding staffing requirements.

Definition 2.4 (Break)**.** A *break B* is a set of consecutive 0-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 2.5 (Work period)**.** A *work period W* is a set of consecutive 1-slots and the succeeding $0'$-slot.

Definition 2.6 (Breaktime)**.** The *breaktime* for a shift S is the number of 0-slots that have to be assigned to S. The breaktime depends on the shift's length $|S|$ and is given as input by a function $\tau(|S|)$.

Definition 2.7 (Temporal Constraint)**.** A *temporal constraint* defines global restrictions on the lengths and locations of breaks and worktime in shifts.

Definition 2.8 (Break pattern)**.** A *break pattern D_s* for shift S, $D_s \subset S$ is a set of timeslots defining an assignment of breaks satisfying the shift's breaktime $\tau(|S|)$ and the set of constraints $\mathcal{C}$. It holds that $|D_s| = \tau(|S|)$.

Definition 2.9 (Domain)**.** The *domain $\mathcal{D}_s$* of a shift S is the set of all possible break patterns for S. $\mathcal{D}_S$ is induced by $\mathcal{C}$ and $\tau(|S|)$. The size of $\mathcal{D}_S$ usually grows exponentially with respect to $|S|$.

Definition 2.10 (Staffing requirements)**.** *Staffing requirements* denote the number of required 1-slots for each timeslot $t \in T$.

We formally define BSP as follows:

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

k: Number of timeslots. Given k, we define a set of consecutive timeslots $T = \{1, 2, ..., k\}$.

$\mathcal{S}$: Collection of shifts, each shift taking place within T, $\forall S \in \mathcal{S} : S \subseteq T$.

$\tau(|S|)$; Function that maps each shift length to a value denoting its breaktime.

$\rho(t)$: Function that maps each timeslot t to its staffing requirements.

$\mathcal{C}$: Set of temporal constraints $\{C_1, C_2, ..C_5\}$, as defined below.

C_1 **Break positions:** Defined by (d_1, d_2). In each shift, the first d_1 and the last d_2 slots must be 1-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.

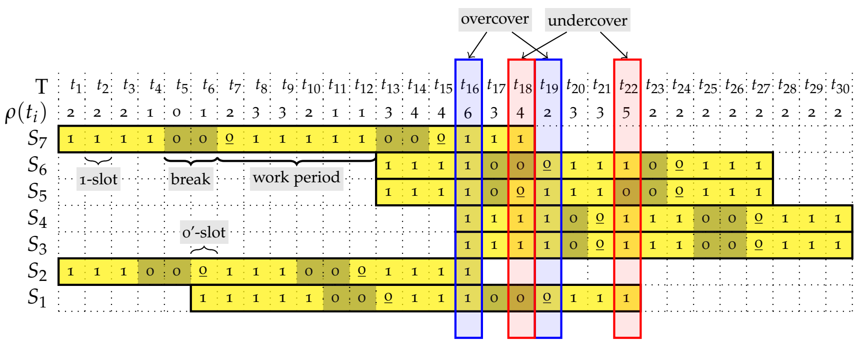
C_2 **Lunch breaks:** Defined by (h, g, l_1, l_2). Each shift S with $|S| > h$ must contain a lunch break consisting of at least g consecutive 0-slots. The break may start earliest l_1 timeslots and end latest l_2 timeslots after the start of its associated shift.

C_3 **Duration of work periods:** Defined by (w_1, w_2). The number of timeslots in each work period must range between w_1 and w_2.

C_4 **Minimum break times:** Defined by (w, b). A work period containing a number of timeslots greater or equal w must be followed by a break greater or equal b timeslots.

C_5 **Break durations:** Defined by (b_1, b_2) The length of each break must range between b_1 and b_2 timeslots.

A sample instance of BSP with a possible solution is depicted in Figure 2.1.



Solution 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, $C_1 = (3,3)$, $C_2 = (25,4,7,7)$, $C_3 = (3,6)$, $C_4 = (5,2)$, $C_5 = (1,3)$

Figure 2.1: A sample instance of BSP with solution and explanations

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 $D \in \mathcal{D}_S$, such that the following objective function $F(\mathcal{B}, \mathcal{P})$ is minimised:

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

where

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

2.2 COMPLEXITY

We present a proof of NP-completeness for Bsp under the condition that break patterns are given explicitly as part of the input. The hardness is proven by reduction from the well-known NP-complete problem Set Cover by 3-Sets, abbreviated X3C [17].

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

We show that Bsp is in NP.

Proof. We are given input $(k, \mathcal{S}, \tau, \rho, \mathcal{C})$ and construct a certificate by generating a break pattern for each $S \in \mathcal{S}$. The certificate can be checked in polynomial time, or more precisely in $O(k * |\mathcal{S}|)$, by checking for each shift if the temporal constraints are satisfied, and then checking for each timeslot if the staffing requirements are satisfied. □

We define Bsp′ as modification of Bsp as follows:

Definition 2.11 (Bsp′). $0'$-slots are eliminated from the problem, i.e. break patterns in Bsp′ contain only 0- and 1-slots.

As *instance* we are given a tuple $(k, \mathcal{S}, \tau, \rho, \gamma)$, where the definition of $k, \mathcal{S}, \tau, \rho$ equal those in Bsp and $\gamma(S)$ is a function that maps each shift S to a set $\mathcal{D}_s$ of break patterns. It must hold that $\forall D \in \mathcal{D}_s : |D| = \tau(S)$.

The *objective* of Bsp′ is to decide whether for an instance $\mathcal{P}'$ there exists a solution $\mathcal{B}$ that relates each shift $S \in \mathcal{S}$ to a break pattern $D \in \mathcal{D}_s$ such that $F(\mathcal{B}, \mathcal{P}') = 0$.

The difference between Bsp and Bsp′ is that for Bsp the set of possible break patterns, or domain, $\mathcal{D}_s$ for a shift S is given implicitly by $\tau(S)$ and $\mathcal{C}$ while for Bsp′ this set is given explicitly by $\gamma(S)$.

We show that Bsp′ is NP-complete.

Proof. The NP-membership of Bsp′ follows from the NP-membership for Bsp .

We show that Bsp′ is NP-hard by reduction from the well-known NP-complete problem X3C [17].

Definition 2.12 (X3C)**.** *Instance*: A set U with $|U| = 3m$, a collection $\mathcal{F}$ of 3-element subsets of U, $\forall F \in \mathcal{F} : |F| = 3$. *Question:* Does $\mathcal{F}$ contain an exact cover for U, i.e. a subcollection $\mathcal{F}' \subseteq \mathcal{F}$, s.t. each element of U occurs in exactly one member of $\mathcal{F}'$?

From an instance of X3C with input $(U, \mathcal{F})$ we construct an instance of Bsp′ with input $(k, \mathcal{S}, \tau, \rho, \gamma)$ as follows:

- $k = |U|$
- $|\mathcal{S}| = \frac{k}{3}$
- $\forall\, S \in \mathcal{S} : S = T$ (remember that $T = 1, 2, ..., k$ as per definition of Bsp′).
- $\forall\, S \in \mathcal{S} : \tau S = 3$.
- We define a bijective function σ that enumerates all elements $u_i \in U$: $\sigma(u_1) = 1, \sigma(u_2) = 2, ..., \sigma(u_{3m}) = 3m$.
- $\forall\, t \in T : \rho(t) = |\mathcal{S}| - 1$, or in other words, in each timeslot exactly one break is required.
- Let $\sigma(F) = \{\sigma(f) \mid f \in F\}$, then for each shift $S \in \mathcal{S}$, $\gamma(S) = \{\sigma(F) \mid F \in \mathcal{F}\}$

Obviously, all of these steps can be done in polynomial time.

We now show the equivalence of this reduction.

Let $\mathcal{P}'$ be an instance of the Bsp′ problem obtained from an instance $\mathcal{X}$ of the X3C problem by applying the steps described above.

With this construction, the number of shifts in $\mathcal{P}'$ equals the number of sets in $\mathcal{X}$ that are needed to cover U. All shifts are of the same length, stretching over all timeslots. The timeslots are defined by applying σ to all elements in U. In each timeslot, exactly one 0-slot is required. All shifts $S \in \mathcal{S}$ share the same domain $\mathcal{D}_s$, which contains the contents of $F \in \mathcal{F}$ mapped by σ.

We now show that $\mathcal{X} \in$ X3C $\Leftrightarrow \mathcal{P}' \in$ BSP$'$ holds.

$\Rightarrow$ If $\mathcal{X}$ is a YES instance of X3C, then there exists a collection of sets $\mathcal{F}'$ such that all elements $u \in U$ occur in one $F \in \mathcal{F}'$. By applying σ to all elements in each set $F \in \mathcal{F}'$, we obtain a solution for $\mathcal{P}'$, namely a break pattern for each shift. Thus, $\mathcal{P}'$ is a YES instance of the reduced Bsp$'$ problem.

$\Leftarrow$ If $\mathcal{P}'$ is a YES instance of Bsp$'$, then there exists a set of break patterns such that each timeslot $t \in T$ occurs in exactly one break pattern (each timeslot must contain exactly one 0-slot according to ρ). By applying σ^{-1} to all elements in all sets in $\mathcal{B}$, we obtain a solution for $\mathcal{X}$. Thus, $\mathcal{X}$ is a YES instance of the original X3C problem. □

We present an example of the X3C to Bsp$'$ reduction as follows:

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

- $k = |\mathcal{U}| = 9$, thus $T = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $|\mathcal{S}| = 3$
- $S_1 = S_2 = S_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- σ

$\mathcal{U}$	A	B	C	D	E	F	G	H	I
T	1	2	3	4	5	6	7	8	9

-

t	1	2	3	4	5	6	7	8	9
$\rho(t)$	2	2	2	2	2	2	2	2	2

- $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = (\{1, 3, 6\}, \{3, 4, 5\}, \{6, 7, 8\}, \{1, 4, 7\}, \{2, 7, 9\}, \{4, 5, 8\})$

A solution for $\mathcal{P}'$ is the mapping $(S_1, \{1, 3, 6\}), (S_2, \{2, 7, 9\}), (S_3, \{4, 5, 8\})$. Figure 2.2 depicts this solution. It is easy to construct the solution for the X3C instance $\mathcal{X}$ by looking up the values in the σ function.

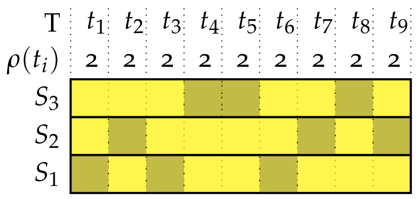


Figure 2.2: Solution for Bsp′ instance reduced from X3C instance

2.3 RELATED WORK

In literature, break scheduling problems have been addressed in two ways: As part of the so-called shift scheduling problem and, as in our case, as break scheduling problem with fixed shifts. The objective of the former problem is to schedule shifts and breaks at the same time according to given constraints, while for the latter, readily scheduled shifts are part of the input and the objective is to place breaks into the shifts according to given constraints.

In practice, the decision on whether to schedule shifts along with breaks or each independently depends on many different factors, such as predictability of staffing requirements or constraints imposed by federal laws. It makes sense to schedule shifts and breaks independently in if employer has to announce shiftplans well in advance with little possibility to change, while staffing requirements are difficult to predict and change rapidly between short timeslots. In this case, new break assignments may be necessary for shifts that cannot be moved.

Another point is that, to our best knowledge, there has not yet been any work on shift scheduling with breaks for problem formulations that allow an arbitrary number of breaks. To find a solution for such a problem, one way is to schedule shifts together with breaks at the same time and another to first schedule the shifts and then to insert the breaks into the existing shiftplan. It is unclear which approach is more effective, but the first one certainly considers a larger search space.

Most work however can be found on shift scheduling with breaks, though in all problem formulations the number of breaks is restricted. We give a quick overview on related work on both the shift scheduling with breaks problem and the stand-alone break scheduling problem.

2.3.1 *Shift scheduling with breaks*

Research on methods to solve or optimise shift scheduling problems with breaks started in the 1950's with Dantzig's set-covering formulation [12] for an assignment of shifts with up to three breaks for toll booth staff. In this formulation feasible shifts are enumerated based on possible shift starts, shift lengths, breaks, and time windows for breaks. In the 70's, this problem was addressed by Segal [30] who presented a set of network-flow formulations for operator-scheduling, where shifts with breaks and reliefs for telephone operators were to be scheduled.

Based on Dantzig's formulation, various integer programming approaches were introduced in the 1990's. Bechtold and Jacobs proposed an integer programming formulation [4] where break assignments are modelled implicitly. The advantage of this approach is that the number of variables does not increase as rapidly as within Dantzig's formulation. They claimed this method to be superior to Dantzig's formulation with respect to execution time, memory requirements and finding optimal solutions for larger instances. Thompson [33] presented an integer programming model with implicit modelling of both shifts and breaks. He reported improvements on execution time compared to Bechtold and Jacobs. Yet another integer programming model is proposed in Aykin [3]. Aykin later compared the different models for shift scheduling with breaks in [2]. While [3] requires more variables than [4], their approach finds optimal solutions in shorter time.

A very recent work from Rekik et al. [29] in 2010 addresses a problem more similar to our problem formulation. They propose an implicit model for a shift scheduling problem with multiple breaks. Additionally one break in each shift can be fractioned into multiple breaks resulting in break profiles, which are something similar to what we denote as break patterns. Additionally, possible work stretches, similar to our work periods, are constrained by a maximum and minimum length.

Besides exact methods, also meta-heuristic methods have been proposed for shift scheduling problems with breaks. Tabu search algorithms have been introduced by both Canon [11] and Tellier and White [32] for problems in the call or contact center industry. Both report satisfactory results. A genetic algorithm with local search and parametrised fitness function was presented by Yamada et al. [35] for an information operator scheduling problem. The shifts according to their problem definition last up to five hours and contain up to five breaks.

2.3.2 *Break scheduling*

The break scheduling problem with a fixed shiftplan has been attended by Beer et al. [5, 6]. They propose a min-conflicts based local search algorithm, a tabu search and a simulating annealing approach to optimise this problem. The min-conflicts heuristic iteratively improves the current solution by focusing only on breaks causing temporal constraint or staffing requirements violations. In each iteration, a break that violates a constraint is selected randomly and its neighbourhood constructed. The move leading to a solution improving or at least not worsening the current violation degree is performed and the new solution used for the following iteration.

To avoid local optima the authors apply a random walk strategy: In each iteration a random, possibly worsening, move is performed on a randomly selected break with some probability.

The authors evaluate different parameter settings and publish their best results for two different flavours of the algorithm, one considering temporal constraints as hard constraints and thus resolving them from the very beginning, and another one which randomly initialises solutions which may violate temporal constraints. The second version resolves temporal constraints along with the staffing requirement violations.

3 MEMETICS

The concept of memes was presented by evolutionary biologist Richard Dawkins in the last chapter of his book "The Selfish Gene" in 1976 [14]. His intention was to show applications of Darwin's theory of evolution [13] to other fields than biological evolution. He claims that cultural development within human societies undergoes a process that can also be seen as evolutionary. His ideas were picked up by scientists of different disciplines such as psychologist Susan Blackmore [7], philosopher Daniel Dennet [16] or anthropologist Scott Atran [1], just to mention a few. We take advantage of the concept of memetics to design an optimisation algorithm, as suggested by Moscato [23].

3.1 CONCEPT AND TERMINOLOGY

In this work, we will stick to the terminology coined by Dawkins and Blackmore [7, 14]. Similarly to genes, *memes* represent replicable units that are hosted by an *individual*. However, rather than for a biological unit, a meme stands for a cultural unit like a thought, an idea, a piece of music. Multiple memes may work together and make up cultural entities like music genres, political or religious ideologies, or instructions on how to construct something. Other than genes, memes are of non-physical, intangible nature. A set of memes with different characteristics is referred to as *memepool*.

Looking at the development of cultural entities, a behaviour similar to biological evolution can be observed: Cultural entities, or sometimes only single memes, are passed on within individuals through communication, i.e. they *replicate*. Through misunderstandings or misinterpretations they may change arbitrarily, i.e. they *mutate*. *Selection* might take place through personal preference of an individual for a specific meme or by external influence, for example by censorship. An important difference between genes and memes is that the latter can be improved independently by its hosting individual, e.g. a person can improve an idea previously copied from someone else.

Based on the work of Dawkins and Blackmore [7, 14], let us discuss the following hypothetical example: We define the cultural entity "Spanish Tortilla Preparation Instructions" consisting of the memes "Ingredients", "Potato-cutting technique", "Frying temperature". The ingredients are usually potatoes, eggs and olive oil, but sometimes onions are added. Potatoes can be cut as sticks, even or uneven slices, or tetrahedrons. Each of these options represents a meme, which can be manipulated and passed on to fellow individuals.

In our example, suppose that individual A, who used to cut the potatoes as sticks, learns from individual B the technique of tetrahedron-slicing. He likes it and replaces his previously obtained stick method. Later, he publishes this recipe in a bestselling book which causes its 10,000 readers to also apply and spread the tortilla recipe with the recommendation to cut the potatoes as tetrahedrons. The tetrahedron meme has thus successfully superseded the stick meme. Here, *Replication* took place. Other than in biological evolution, memes are passed on also between individuals of the same generation.

Now consider a new individual C copying the tortilla recipe from individual A, but he understands octahedron instead of tetrahedron. Now he has serious difficulties cutting potatoes in that shape, eventually gives up, and forgets about the whole tortilla recipe. The tortilla recipe that includes the octahedron meme thus extincts. It was not fit enough to survive the requirements of tortilla preparation. This incorrectly imitated meme can be seen as *mutated*. Such an incorrect imitation can happen through simple misunderstandings or misinterpretations occuring quite frequently in human communication.

The *selection* mechanism was already implicitly included in the previous two paragraphs. Individual A selected the tetrahedron slicing method over the stick method because it improved his overall tortilla cooking and eating experience. Similarly, the mutated octahedron method was not selected to continue inside the memepool.

As opposed to genes, an individual is able to independently modify any of its memes. Applying such a modification, the individual usually aims at *improving* the cultural entity as a whole. In our example, this could be towards optimising the preparation time and taste of the tortilla. It is easy to see that this optimisation can be quite subjective. While the preparation time is relatively easy to measure, taste is subject to personal preference.

This mechanism of local improvement along with high replication rates is what accelerates the evolution of memes compared to the evolution of genes. In biological evolution, as presented by Darwin [13], genetic changes appear randomly by mutation while selection pressure makes sure genes with a lower fitness extinct. Replication occurs only by heredity, which means a gene is replicated only from one generation to another. This makes the biological evolution process much slower compared to memetic evolution, where memes may spread to an arbitrary number of individuals within or between generations. Changes occur much more frequently, given that each individual may modify its memes in any way it likes. Also memes can improve much faster, since each individual has control over its memes, as opposed to genes, which are improved over many generations by natural selection.

3.2 from memetics to memetic algorithms

The term Memetic Algorithm was introduced by Moscato in [23]. Memetic algorithms are also known as Hybrid Genetic Algorithms as presented by Goldberg [19]. 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, also 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. We can also say that individuals, each carrying a (probably sub-optimal) solution, are able to independently improve their genes. Looking into the concept of memetics as described in the previous section, we find that it describes exactly this behaviour. Further, crossover may take place within more than two individuals and we will thus denote it as "interaction". Selection is often integrated in the interaction process in this way, that an individual, when copying memes from others, may choose memes with specific characteristics over others.

Memetic algorithms have been investigated on many different problems, such as the travelling salesman problem [24], timetabling problems [9, 10] or quadratic assignment problem [21].

The challenge of designing an algorithm based on memetics to solve a particular optimisation problem includes many different aspects. The very basis of any memetic algorithm is an appropriate *memetic representation* of the problem, i.e. the definition of a meme, an individual and a fitness function. Only upon this representation, the interaction, selection, mutation and improvement operators can be built.

Each of these operators again requires careful definition taking into consideration many different options: How exactly should the interaction take place? How many individuals should be involved in each interaction? Regarding which criteria should an meme be chosen over its competing memes? Which kind of mutation should be applied? Should every single meme be locally improved or only selected ones? If we choose for selected memes, how should they be selected? How intensively should the local improvement be performed? Which method should be used for local improvement?

The choice on the local improvement process itself again opens up a broad range of possibilities. In fact, any heuristic, and for some problems also exact methods, can be used. For optimisation problems, local search is a popular choice. However, local search again comes in many different flavours: One or more neighbourhoods can be defined. In the case of various neighbourhoods, many different exploration strategies exist. And within the chosen strategy, should we use best, random or first improving steps? Integrate a tabu list? Use perturbation?

Additionally, several parameters might influence the algorithm, which impose an additional level of difficulty for the design of a good memetic algorithm. Examples for such parameters are the number of individuals, i.e. the population size, the intensity of local improvement or the selection pressure.

We address these questions in Chapter 4 within the description of the algorithms, and in Chapter 5 within the discussion of the parameter evaluation.

4 SOLVING THE BREAK SCHEDULING PROBLEM

To solve the Break Scheduling Problem, we propose two different memetic representations, one defining a meme as a single shift and the other defining a meme as a group of shifts which largely overlap in time. Upon these representations, we present three different novel methods to obtain good solutions for BSP. The algorithms are based on the concepts discussed in Section 3. In Chapter 5 we show how the different representations influence the quality of the solutions.
The following characteristics all algorithms have in common:

- Initialisation by random assignment of break patterns followed by a simple local search.
- Use of elitism, i.e. the fittest individual (*elitist*) is determined after each operation and is prevented from worsening its fitness in the following iteration.
- Creation of new individuals by copying memes out of the current memepool.
- Mutation of memes by randomly changing break assignments.
- Local improvement of memes by a local search heuristic.
- The local search is based upon the same set of three neighbourhoods.

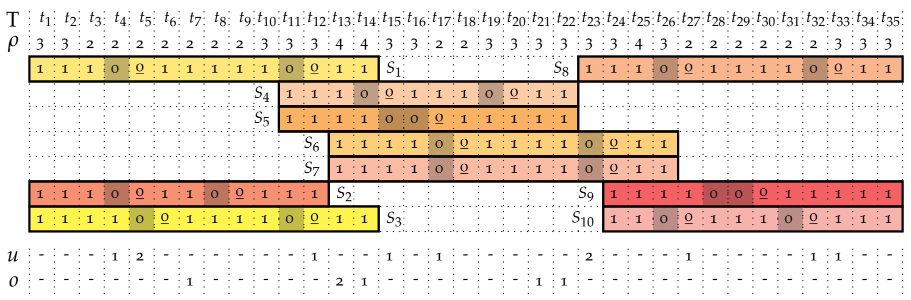
Each of these operations can be done in different ways depending on the memetic representation in use. We propose one algorithm, Memetic Algorithm with Representation 1 or MAR1, based on the first representation, and two algorithms, Memetic Algorithm with Representation 2 or MAR2 and Memetic Algorithm with Penalty System or MAPS, based on the second representation. MAPS, as the name suggests, additionally uses a penalty system that tries to escape local optima.

We first introduce the different representations and some terminology, then describe the initialisation procedure, the neighbourhoods and the local search procedure. Each of the algorithms with its specific operators is then described separately.

4.1 REPRESENTATIONS AND DEFINITIONS

By memetic representation we refer to the representation of a solution $\mathcal{B}$ by a set of memes $\mathcal{M}$, similarly to genetic representations in genetic algorithms. The genetic operators are defined upon this representation. We present two different representations.

Definition 4.1 (Memetic Representation 1)**.** A meme M is represented by exactly one shift $S \in \mathcal{S}$ and its associated breaks according to $\mathcal{B}$ and each shift represents a meme, i.e. $M_1 \cup M_2 \cup \cdots \cup M_m = \mathcal{S}$



Shiftplan with solution represented by set of memes $\mathcal{M}$ with 10 memes, each meme represented by one shift.

Figure 4.1: Memetic Representation 1

The problem with Representation 1 is a strong interference between memes regarding the satisfaction of staffing requirements. This makes the design of effective genetic operators difficult, as we will see in Section 4.6. We propose a second memetic representation, which aims at avoiding these interferences.

Memetic Representation 2 overcomes this problem by regarding interfering shifts as memes. Shifts interfere if they take place during the same time period and thus may assign breaks to the same timeslots.

Definition 4.2 (Memetic Representation 2)**.** A meme M is defined by a period $[m', m'')$, with $m', m'' \in T$ and contains

- Those and only those shifts $S \in \mathcal{S} : m' \leq \lfloor (S_e + S_s)/2 \rfloor \leq m''$, S_s and S_e denoting shift start and end

- The breaks associated to these shifts

Each shift $S \in \mathcal{S}$ is thus contained in exactly one meme: $\forall M_i, M_j \in \mathcal{M} : M_i \cap M_j = \emptyset, M_1 \cup M_2 \cup \dots \cup M_m = \mathcal{S}$.

We use the following heuristic to determine the periods that induce the set of memes for Representation 2: For each timeslot $t \in T$ let 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 penalty 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. These values are described in more detail in Section 2.1.

The lower the penalty value $p(t)$ for a timeslot t, the less breaks can be assigned to t. This makes it a good separation point.

We determine a set of timeslots $T' \subset T$ with size $|T'|$ given by a parameter such that

- $\sum_{t' \in T'} p(t')$ 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 all pairs 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 $t'_0 = t$ s.t. $\min_{t \in T} p(t)$, i.e. the timeslot with the lowest penalty value, ties are broken randomly. We continue by adding timeslots $t'_i = t$ s.t.

- $\min_{t \in T \setminus T'} p(t)$
- $\forall t'_k, t'_j \in T' \cup t : \left| t'_k - t'_j \right| > d$

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 $[t'_i, t'_{i+1})$.

Definition 4.3 (Meme Fitness)**.** For Representation 2, 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 $T' = \bigcup S, \forall S \in M$

$$F(M) = F(\mathcal{B}, \mathcal{P}, T') = w_o \cdot O(\mathcal{B}, \mathcal{P}, T') + w_u \cdot U(\mathcal{B}, \mathcal{P}, T')$$

where

- w_o and w_u are the same weights for over- and undercover defined in Section 2.1.
- $O(\mathcal{B}, \mathcal{P}, T') = \sum_{t \in T'} \max(0, \rho(t) - \omega(\mathcal{B}, t))$ i.e. sum of undercover
- $U(\mathcal{B}, \mathcal{P}T') = \sum_{t \in T'} \max(0, \omega(\mathcal{B}, t) - \rho(t))$ i.e. sum of overcover
- $\omega(\mathcal{B}, t)$ the number of 1-slots in timeslot $t \in T'$ according to $\mathcal{B}$

This is the fitness function presented in Section 2.1, but applied only on a subset of T.

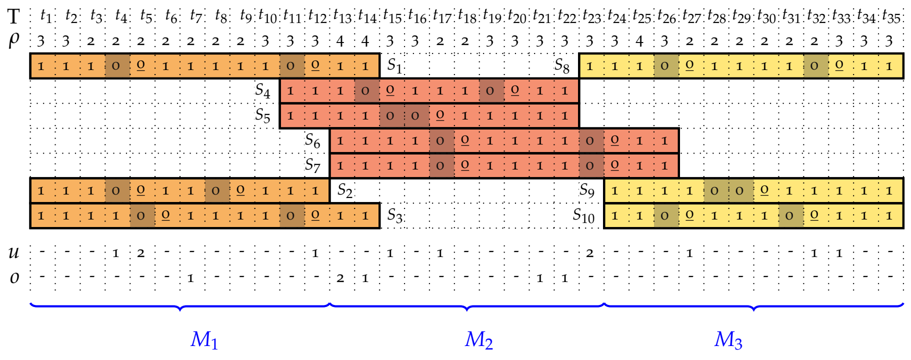
Figure 4.1 depicts a possible memetic representation of BSP.

Definition 4.4 (Individual)**.** An *individual* I contains a solution $\mathcal{B}$, i.e. a mapping of shifts to break patterns for each shift $S \in \mathcal{S}$ and a *Fitness value* $F(I)$, which is the value of the objective function $F(\mathcal{B}, \mathcal{P})$.

Definition 4.5 (Population)**.** A *population* is a set $\mathcal{I}$ of individuals.

Definition 4.6 (Generation)**.** A *generation* is a population during an iteration of the algorithm.

Definition 4.7 (Memepool)**.** A *memepool* is the set of all memes in a given generation.



Shiftplan with 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 $M_3 = (\{S_8, S_9, S_{10}\}, m' = 24, m'' = 35)$

Figure 4.2: Memetic Representation 2

4.2 BREAK PATTERNS

As mentioned in Section 2.1, using input data $\mathcal{C}$ and τ, a set of break patterns, or domain, $\mathcal{D}_s$ for each shift S can be computed. We refer to $\mathcal{D}_s$ as *domain* for shift S.

The problem of finding a single break pattern $D \in \mathcal{D}_s$ can be modeled as simple temporal problem as presented by Dechter et al. [15] and consequently be solved in cubic time with respect to $|S|$ using Floyd-Warshall's shortest path algorithm [26]. The size of a domain $|\mathcal{D}_s|$ usually grows exponentially with respect to $|S|$.

We precalculate a subset $\mathcal{D}'_s$ of $\mathcal{D}_s$ for each shift length with the following restrictions:

- $b_1 = 2$, $b_2 = 3$, b_1, b_2 denoting minimal and maximal allowed length of breaks respectively, except for lunch breaks, which are all set to their minimal size $g = 6$
- Break patterns may include breaks of different sizes. E.g. if $\tau(|S|) = 10$ without lunchbreak, then this breaktime can be made up out of two breaks with length 2 and two breaks with length 3 or five breaks with length 2. Possible break patterns in $\mathcal{D}_s$ for a shift S thus include all possible combinations of break lengths that sum up to $\tau(|S|)$ as well as their permutations. In the example defined 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 $(2,2,2,2,2)$.

$\lvert S\rvert$	$\lvert\mathcal{D}'_s\rvert$	$\lvert S\rvert$	$\lvert\mathcal{D}'_s\rvert$
60	2,514	110	1,028,879
66	7,457	114	7,685,738
72	13,175	116	7,075,413
92	14,149	126	56,150,948
96	223,315	134	181,462,430
102	131,698	138	374,959,311
106	1,475,034	144	1,133,795,593
108	1,283,700		

Table 4.1: Sizes of $\mathcal{D}'$ for different shift lengths

To save computation time, we select only one permutation of each combination at random. For the example $\tau(|S|) = 10$, we would compute all possible patterns with break ordering $(2,2,2,2,2)$ and one selected randomly out of all permutations of $(3,3,2,2)$.

Figure 4.3 depicts an example for different break patterns and Table 4.1 shows the sizes of $\mathcal{D}'_s$ calculated according to the described restrictions for different shift lengths occuring in the publicly available instances. Specific information on the instances is given in Section 5.2.2.

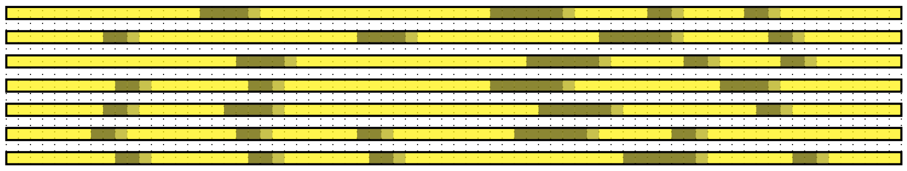
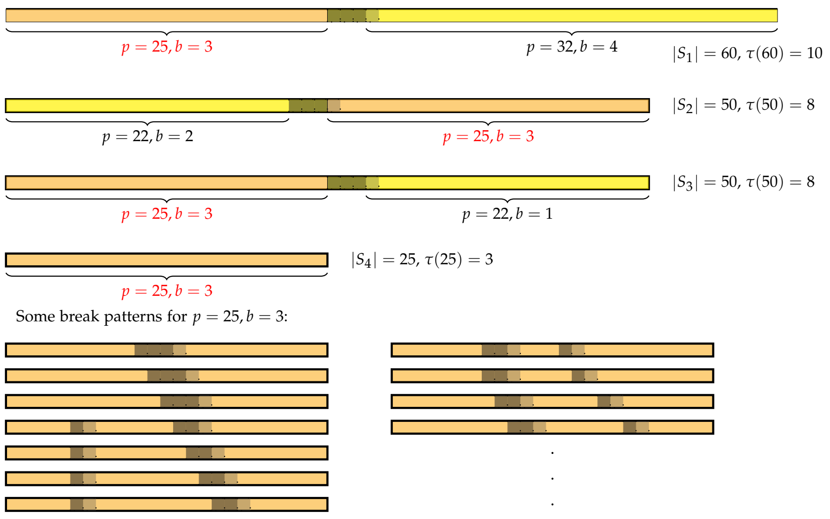


Figure 4.3: 7 possible break patterns for length $|S| = 74$

To save computation time and space, we use the following method to compute $\mathcal{D}'_s$ for each shift: If, according to constraint C_2 (see Section 2.1), 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 a period before and a period after each lunch break position. A domain for any of the periods depends on the length p of the period that is covered, the breaktime b it contains and $\mathcal{C}$. Since $\mathcal{C}$ is defined globally, a domain is identified by (p, b). For each possible period before/after lunch breaks, and periods covering shifts that do not contain a lunch break, we compute a subdomain. Note that periods with the same length and breaktime, i.e. (p, b), share the same domain. We thus compute a subdomain for each tuple (p, b) and for each shift length we store only references to the domains.

An example for this method is depicted in Figure 4.2



$p = 25, b = 3$
$p = 32, b = 4$
$|S_1| = 60, \tau(60) = 10$
$|S_2| = 50, \tau(50) = 8$
$p = 22, b = 2$
$p = 25, b = 3$
$|S_3| = 50, \tau(50) = 8$
$p = 25, b = 3$
$p = 22, b = 1$
$|S_4| = 25, \tau(25) = 3$
$p = 25, b = 3$
Some break patterns for $p = 25, b = 3$:

The first three shifts are divided into two parts by their lunch break. The parts with the same b and p, e.g. $p = 25$ and $b = 3$, as well as S_4 with length 25 and no lunch break, have the same set of break patterns. We thus have to compute and store it only once.

Figure 4.4: Computation of break patterns

4.3 INITIALISATION

We use the same initialisation procedure for all algorithms presented in this thesis.

Each individual I 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 randomised local search procedure is executed on the solution.

The randomised local search iterates the following steps: First, a break b is picked at random out of all breaks of individual I and the set $\mathcal{N}$ of neighbours in its *single assignment* neighbourhood computed. This neighbourhood is described in Section 4.4.

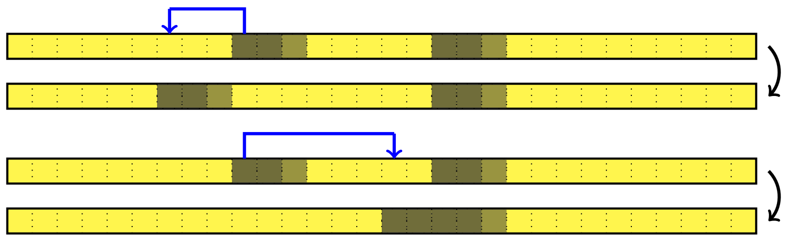
Second, for each $N \in \mathcal{N}$ let $\delta(N, I) = F(N) - F(I)$, i.e. the difference between the fitness values. Let $\mathcal{N}' = \{N \in \mathcal{N} : \delta(N, I) \leq 0\}$. If $|\mathcal{N}'| > 0$ then we pick a $N \in \mathcal{N}'$ at random, otherwise we do nothing.

The local search terminates when for 10 subsequent iterations $|\mathcal{N}'| = 0$, i.e. no improvements could be found.

4.4 NEIGHBOURHOODS

4.4.1 *Single Assignment*

This neighbourhood, $\mathcal{N}_1$, comprises the set of all solutions that are reached by applying a single assignment move. This 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. Examples for single assignment moves are depicted in Figure 4.5. For the instances used in this work (for details see Section 5.2.2), the size of this neighbourhood averages three to four neighbours.

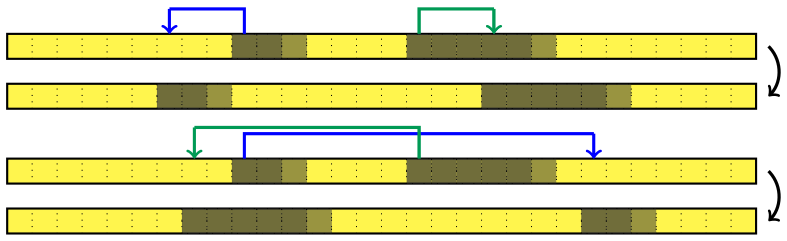


Two examples for possible moves in the single assignment neighbourhood. The second move shows two breaks being joined to a longer break and thus eliminating one $0'$-slot.

Figure 4.5: Single assignment moves

4.4.2 *Double Assignment*

This neighbourhood, $\mathcal{N}_2$, consists of the set of all solutions that are reached by a *double assignment* move. This 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. This neighbourhood is significantly larger then the neighbourhood constructed by single assignment moves. For the instances tested, its size amounts to up to 100 neighbours. This neighbourhood is illustrated in Figure 4.6.

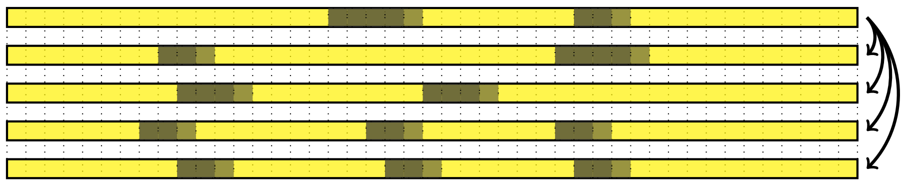


Two examples for possible moves in the double assignment neighbourhood with two breaks of different lengths.

Figure 4.6: Double assignment moves

4.4.3 *Shift Assignment*

This neighbourhood, $\mathcal{N}_3$, consists of a set of solutions that are reached by assigning all breaks associated to the shift S containing b. Possible re-assignments are retrieved from in the pre-calculated set of break patterns $\mathcal{D}'_s$ described in Section 4.2. For performance reasons, we do not consider the complete $\mathcal{D}'_s$, but only a randomly selected subset. The size of this subset determines the size of the neighourhood. Figure 4.7 depicts some shift assignment moves.



Four possible moves in the shift assignment neighbourhood. Any break assignment that satisfies $\mathcal{C}$ can be contained in this neighbourhood.

Figure 4.7: Shift assignment moves

4.5 LOCAL SEARCH

We propose the following local search heuristic, which is used by all three algorithms. 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 a set B of breaks. B may comprise all breaks that are currently included in the individual's solution or only a subset. Second, a neighbourhood $\mathcal{N}$ out of three neighbourhoods $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ is chosen at random with a different probability for each neighbourhood given by 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. 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 $N \in \mathcal{N}'$ s.t. $\min\limits_{N \in \mathcal{N}'} \delta(N, I)$, i.e. the best neighbour is chosen with ties broken randomly. Otherwise we do nothing. 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 of the different neighbourhoods η. Different values for these parameters have been tested for each algorithm. The results are presented in Chapter 5. Algorithm 1 outlines the local search procedure.

4.6 MAR1 – MEMETIC ALGORITHM WITH REPRESENTATION 1

This algorithm is based on Representation 1. The algorithm creates each offspring either by mutation or by crossover from the previous generation. A k-tournament

Algorithm 1 Search (Individual I, Breaks$\{\}B$)

```
1:  $c \leftarrow 0$
2:  **repeat**
3:      $b \leftarrow$ select break $b \in B$ randomly
4:      $\mathcal{N} \leftarrow$ select and compute one of $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$
5:      $\mathcal{N}' \leftarrow \{N \in \mathcal{N} : \delta(N, I) \leq 0\}$
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** $I$
```

selector [8] decides which inidividuals survive in each iteration. The local search is applied in each iteration on a subset of the population. 4.6 outlines this algorithm.

4.6.1 *Selection*

The selection operator selects a set of individuals that survive the current iteration. First, an elitist E is determined by selecting the individual with the best fitness value in the current generation (ties are broken randomly). The elitist individual remains unaltered during the whole iteration.

Second, individuals are selected out of the current generation by a k-tournament selector [8]: This operator randomly takes k individuals, $k \leq |\mathcal{I}|$ out of the current population 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 ($|\mathcal{I}| - 1$ since one individual is represented by the elitist). The population now consists of the elitist and the individuals selected by the tournament selection operator.

4.6.2 *Crossover and Mutation*

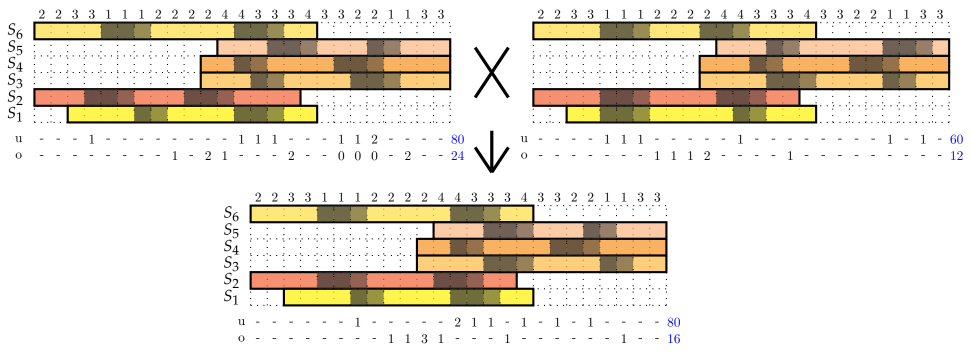
Each individual $I \in \mathcal{I} \setminus E$ is replaced by an offspring created either by mutation or by crossover. The elitist individual remains the same. Crossover takes place with probability α and mutation with $1 - \alpha$, where α is a parameter set experimentally

Algorithm 2 Memetic Algorithm with Representation 1

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

as described in Section 4.5. We propose a combination of two different crossovers. Both select a partner individual J different to I randomly out of the generation and create an offspring inheriting entire shifts (memes) with their breaks from one of the parents. They differ in the decision about which shifts are taken from which partner.

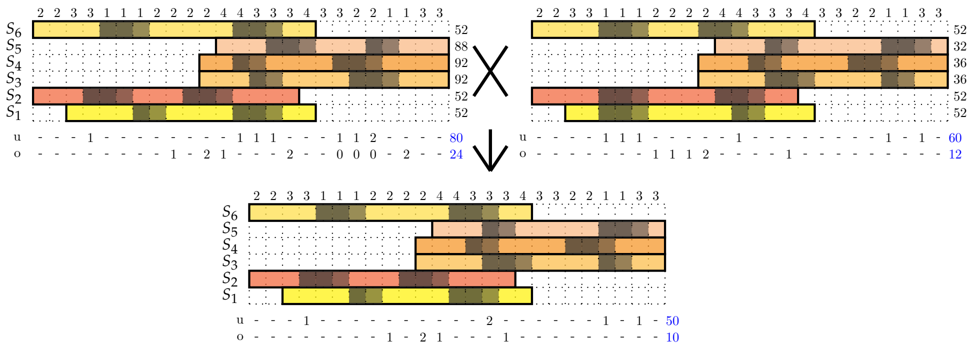
SIMPLE CROSSOVER An offspring is produced by randomly inheriting a percentage γ of all shifts with their assigned breaks from one parent's exchangeable shifts, and the remaining shifts from the other parent. By exchangeable shifts we refer to shifts with equal shift start and length. Figure 4.6.2 depicts this operator. Obviously, the Simple Crossover merely perturbates a solution and improvements are not very likely since the staffing requirement violations are not considered at all.



Two parent individuals and one offspring with under- and overcover violations, assuming undercover weight $w_u = 10$ and overcover weight $w_o = 2$: fitness of left parent 104, fitness of right parent 72 and fitness of offspring $F(\mathcal{B}_3, T) = 96$

Figure 4.8: Simple Crossover

SMART CROSSOVER This crossover looks at each shift of one of the parents and determines the constraint violations the shift is involved in, i.e. the sum of constraint violations of the timeslots covered by the shift. The offspring inherits the respective shift from the parent with lower constraint violations during the timeslots covered by the shift. This way we are more likely to improve the fitness value of the offspring compared to its parents. Figure 4.9 depicts this operator.


2 2 3 3 1 1 1 2 2 2 2 4 4 3 3 3 4 3 3 2 2 1 1 3 3
S_6 S_5 S_4 S_3 S_2 S_1
52 88 92 92 52 52
u - - - 1 - - - - - - - - - 1 1 1 - - - 1 1 2 - - - - 80
o - - - - - - - - - 1 - 2 1 - - - 2 - - 0 0 0 - 2 - - 24
2 2 3 3 1 1 1 2 2 2 2 4 4 3 3 3 4 3 3 2 2 1 1 3 3
52 32 36 36 52 52
u - - - - 1 1 1 - - - - - 1 - - - - - - - - - 1 - 1 - 60
o - - - - - - - - 1 1 1 2 - - - - 1 - - - - - - - - - - 12
2 2 3 3 1 1 1 2 2 2 2 4 4 3 3 3 4 3 3 2 2 1 1 3 3
u - - - 1 - - - - - - - - - - 2 - - - - - - 1 - 1 - 50
o - - - - - - - - 1 - 2 1 - - - 1 - - - - - - - - - 10

Two parent individuals with violations of each shift and under- and overcover violations, assuming undercover weight $w_u = 10$ and overcover weight $w_o = 2$: Fitness of left parent 104, fitness of right parent 72 and fitness of offspring 64. The shifts with lower violations are inherited to the offspring, ties broken randomly. Here, S_1, S_2, S_6 are inherited from the left parent, and S_3, S_4, S_5 from the right parent. The result is an offspring with a better fitness value than both parents.

Figure 4.9: Smart Crossover

MUTATION The mutation operator performs one random move on the given individual using the *single assignment* neighbourhood, which is described in the Section 4.4.

4.6.3 *Local Search*

In each iteration, we select a set $\mathcal{L} \subset \mathcal{I}$ individuals, $|\mathcal{L}| = |I| \cdot \lambda$ of the current population. λ is a parameter whose value has been determined experimentally as described in Section 5.1. On each individual $L \in \mathcal{L}$ the local search heuristic described in Section 4.5 is executed with the set of breaks B comprising all breaks contained in I. Different values tested for λ are described in Section 4.5.

4.7 MAR2 – MEMETIC ALGORITHM WITH REPRESENTATION 2

MAR2 is similar to MAR1, but uses Representation 2. The operators are mostly the same as those presented for MAR1. Within this algorithm, we tested a tabu list as part of the local search heuristic.

Algorithm 3 Memetic Algorithm with Representation 2

```
1:  BUILDDOMAINS
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$(\mathcal{I}) \cup E$
15:     $m \leftarrow |\mathcal{I}| \cdot \lambda$
16:     $\mathcal{L} \leftarrow m$ fittest individuals in $\mathcal{I}$
17:     for all $L \in \mathcal{L}$ do
18:         $B \leftarrow$ all breaks contained in $L$
19:         $L \leftarrow$ SEARCH$(L,B)$
20:     end for
21: until timeout
22: return fittest $I \in \mathcal{I}$
```

4.7.1 *Crossover and Mutation*

The crossover and mutation operators in this algorithm work very similar to those described in Section 4.6.2. Each individual $I \in \mathcal{I} \setminus E$ is replaced by an offspring created either by mutation or by crossover $I \in \mathcal{I}$, except for the elitist individual. Crossover will take place with probability α and mutation with $1 - \alpha$.

The crossover operator selects a partner J different to 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 v to inherit the meme M with better fitness $F(M)$. v is a parameter for which different values are evaluated in Section 5.2.6.

The mutation operator performs one random move on the given individual using the *shift assignment* neighbourhood, which is described in the Section 4.4.

4.7.2 *Selection*

The selection operator for this algorithm is exactly the same as described for the MAR1 algorithm described in Section 4.6.1.

4.7.3 *Local Search with Tabu List*

Like MAR1, MAR2 also performs the local search on a subset of individuals. The search procedure is as described in Section 4.5 with the only exception that the computed neighbourhood $\mathcal{N}$ is reduced by those neighbours that are currently forbidden by a tabu list.

A tabu list [18] 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: The oldest value of L is deleted and the current first slot of b is added. The length $|L|$ of the tabu list thus determines how long a value is kept tabu.

The neighbourhood $\mathcal{N}$ is computed as described in Section 4.5, but reduced by those neighbours that are reached applying a move that includes moving the first slot of b to any of the values contained in L. However, if any of the tabu moves leads to a globally improved neighbour, it is allowed anyway. This is called an aspiration criterion. The tabu list intends to prevent the local search from re-visiting previously computed solutions.

4.8 MAPS – MEMETIC ALGORITHM WITH PENALTY SYSTEM

MAPS uses Representation 2 as does MAR2. However, the algorithm is different in many aspects: The crossover operator uses memes of the whole generation to create an offspring instead of only two parents. We renamed the crossover operator to the more appropriate term "interaction" operator as an analogy to the interaction of multiple individuals in cultural evolution. Further, in this algorithm the selection mechanism is included in the interaction operator. The mutation and local search procedures consider only breaks contained in a subset of an individual's memes. Memes additionally keep a memory to track data about their search history. Algorithm 4 outlines the method.

Algorithm 4 Memetic Algorithm with Penalty System

```
 1: BUILDDOMAINS
 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:         $I \leftarrow$ INTERACT-SELECT($\mathcal{I}$)
 7:         $E \leftarrow$ fittest $I \in \mathcal{I}$
 8:         $\mathcal{M}' \leftarrow$ GET($\mathcal{M}' \subset \mathcal{M}$)
 9:         $I \leftarrow$ MUTATE($I, \mathcal{M}'$)
10:         $B \leftarrow$ all breaks contained in $\mathcal{M}'$
11:         $I \leftarrow$ SEARCH($I, B$)
12:         $I \leftarrow$ PENALTY-UPDATE($I, B$)
13:     end for
14: until timeout
15: return fittest $I \in \mathcal{I}$
```

4.8.1 *Penalty System*

For each meme M we store the following values:

Best fitness value $B(M)$: The best fitness value $F(M)$ the meme reached since the start of the algorithm

Penalty value $P(M)$: Number of iterations since last update of best fitness value

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 interaction 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 which is 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. We describe this behaviour more detailled within the following sections.

4.8.2 *Selection and Interaction*

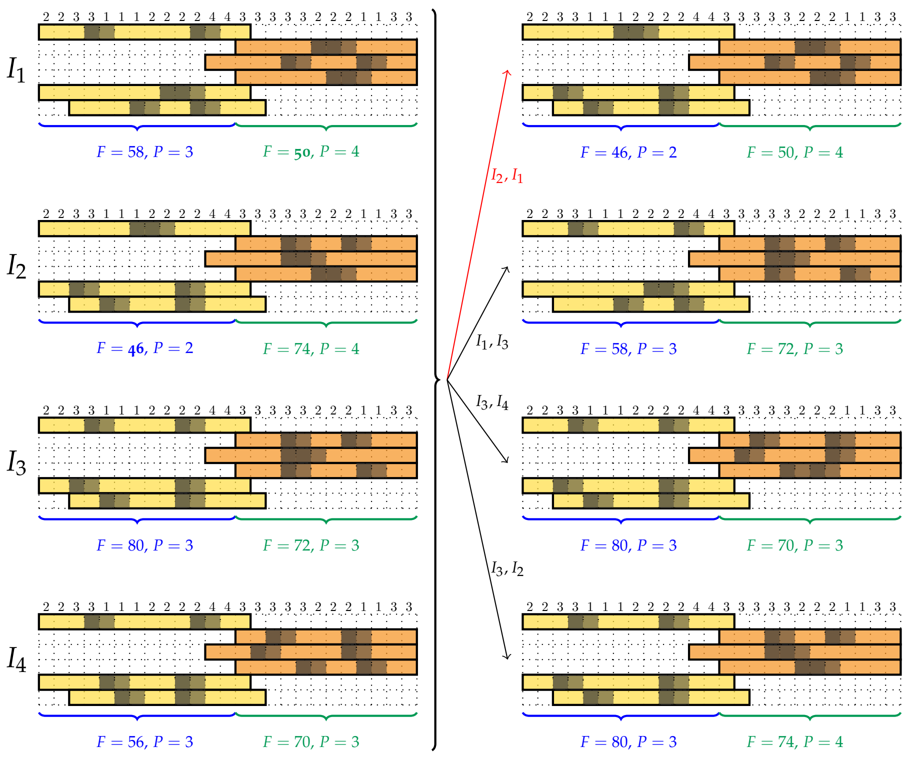
The interaction procedure consists of two parts. We first create an individual by selecting each meme M with the best current fitness value $F(M)$ out of the current memepool. This individual is likely to become the elitist individual in the current population.

The second step contains a selection procedure: Each of $|\mathcal{I}| - 1$ individuals, $|\mathcal{I}|$ being the population size, is created as follows: For each meme M with period $[m', m'')$ we perform a k-tournament selection [8] on the set of memes with the same period in the current memepool: We select k memes with the same period $[m', m'')$ at random out of the current memepool. Out of these k memes, the meme with the lowest penalty value $P(M)$ is inherited to the offspring.

The first part assures to survive the best memes in the current memepool. The second part forms the actual interaction 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 interaction operator as described above. Figure 4.10 depicts the interaction operator.

4.8.3 *Mutation and Local Search*

We evaluate the fitness function $F(I)$ of each individual $I \in \mathcal{I}$ and determine individual $E \in \mathcal{I}$, which is the individual with the lowest value for $F(I)$. On each individual except the elitist $I \in \mathcal{I} \setminus E$ we perform the following procedure:



A population of $|\mathcal{I}| = 4$ individuals create offsprings by interaction. 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 interaction may occur from shifts overlapping into different memes.

Figure 4.10: Interaction operator

For each individual $I \in \mathcal{I}$ we define a set $\mathcal{M}' \in \mathcal{M}$ of memes, such that $\mathcal{M}'$ contains the memes with the lowest penalty values (ties are broken randomly). Only memes in $\mathcal{M}'$ are going to be mutated and locally improved in the current iteration.

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. The size of $\mathcal{S}'$ is a parameter value. Different values for this parameter are evaluated in Section 5.2.7.

The local search is executed as described in Section 4.5 with set B containing the set of breaks contained in $\mathcal{M}'$.

4.8.4 *Penalty Update*

The last step is to update the values for $F(M)$, $B(M)$ and $P(M)$ for each $M \in \mathcal{M}'$: We compute $F(M)$ as described in Section 4.1 and update as follows:

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

The application of each of the methods described in this chapter along with the evaluation of different parameter settings is described in the next chapter.

5 EMPIRICAL PARAMETER EVALUATION

In Chapter 4 some parameters emerged that influence the progress and the results of the algorithm. For instance, a high population size $|\mathcal{I}|$ may provide higher diversity and different starting points for the local search, but on the other hand consume more CPU time. For each algorithm we evaluate a set of parameters, for which a preliminary analysis showed a great impact on the solution qualities. While for MAR1 we only compare average values, for MAR2 and MAPS the impact of each parameter is assessed using statistical methods.

5.1 EVALUATION MAR1

Parameters for MAR1 were evaluated only on the set of publicly available, randomly created instances [31] also used by the authors of [6]. We selected a set of instances for which we executed five runs for each parameter value with a timeout of 900s. The final runs were executed with the parameter values that gave the best results on average solution qualities.

We tested the following parameters:

$|\mathcal{I}|$ Population size

κ Selection pressure: Number of individuals performing a tournament

α Crossover/Mutation: Probability an offspring is created by crossover, $1-\alpha$ probability to an offspring is created by mutation

γ Preference for smart crossover over simple crossover

λ Search rate: Percentage of population the local search is applied on

μ Search intensity: Number of iterations the search continues without finding improvements

(η_1, η_2, η_3): Probability for each neighbourhood to be selected in each local search iteration

$|\mathcal{N}_3|$: Size of the shift assignment neighbourhood

The initial parameters were set after some trial and error preliminary runs. Each parameter was tested after another and the parameters successively adapted according to the result of each experiment. The following were the initial parameter settings: $|\mathcal{I}| = 50$, $\kappa = 3$, $\gamma = 0.8$, $\alpha = 0.7$, $\lambda = 0.15$, $\mu = 700$, $(\eta) = (0.3, 0.3, 0.3)$, $|\mathcal{N}_3| = 1000$.

The following tables describe the average values of each set of runs. The best values are highlighted.

Inst. \ $\lvert\mathcal{I}\rvert$	10	20	30	40	70
ran1-1	1,052	968	**886**	891	1,048
ran1-2	2,738	2,698	2,437	**2,430**	2,578
ran1-5	1,430	1,402	1,200	**1,163**	1,306
ran1-7	1,185	1,146	1,016	**1,009**	1,149
ran2-1	2,100	1,855	**1,778**	1,794	2,043

Table 5.1: MAR1: Population size

Inst. \ κ	2	3
ran1-1	**792**	1,084
ran1-2	**2,500**	3,052
ran1-5	**1,086**	1,629
ran1-7	**934**	1,293
ran2-1	**1,664**	2,492

Table 5.2: MAR1: Selection pressure

Inst. \ γ	1.0	0.0	0.8	0.9
ran1-1	887	1,201	**886**	938
ran1-2	2,657	2,758	**2,437**	2,508
ran1-5	1,268	1,374	1,200	**1,193**
ran1-7	1,020	1,345	**1,016**	1,075
ran2-1	1,880	2,130	**1,778**	1,860

Table 5.3: MAR1: Smart crossover preference γ

Inst. \ α	0.7	0.9
ran1-1	891	**876**
ran1-2	2,430	**2,346**
ran1-5	1,163	**1,071**
ran1-7	**1,009**	1,018
ran2-1	**1,794**	1,830

Table 5.4: MAR1: Crossover probability α

Inst. \ λ	0.1	0.15	0.2
ran1-1	809	**798**	1,012
ran1-2	2,352	**2,307**	2,400
ran1-24	1,359	**1,352**	1,480
ran1-28	3,077	3,042	**3,020**
ran2-4	2,480	**2,413**	2,534

Table 5.5: MAR1: Search rate λ

Inst. \ μ	100	500	700	1,000
ran1-1	959	798	**745**	855
ran1-2	2,447	**2,307**	2,331	2,405
ran1-24	1,376	1,352	**1,230**	1,270
ran1-28	3,594	3,042	**2,964**	3,082
ran2-4	2,480	2,413	**2,378**	2,474

Table 5.6: MAR1: Search intensity μ

Inst. \ η	$(.3, .3, .3)$	$(.8, .15, .05)$	$(.05, .8, .15)$	$(.15, .05, .8)$
ran1-1	670	**654**	675	685
ran1-2	**2,066**	2,174	2,135	2,197
ran1-5	**844**	884	866	872
ran1-7	**712**	799	813	756
ran2-1	**1,232**	1,398	1,328	1,373

Table 5.7: MAR1: Neighbourhoods η

Inst. \ $\lvert\mathcal{N}_3\rvert$	100	300	1000
ran1-1	**631**	705	731
ran1-2	2,262	**2,251**	2,269
ran1-5	972	**920**	981
ran1-7	794	**784**	822
ran2-1	**1,224**	1,454	1,523

Table 5.8: MAR1: Size of the shift assignment neighbourhood $\mathcal{N}_3$

We finally executed the algorithm with the parameter values that which proved to be most appropriate according to the experiments. The parameters were set as follows: $|\mathcal{I}| = 40$, $\kappa = 2$, $\gamma = 0.8$, $\alpha = 0.9$, $\lambda = 0.15$, $\mu = 700$, $\eta = (0.3, 0.3, 0.3)$ and $\mathcal{N}_3 = 300$. Results are compared with those of our other algorithms and those from literature in Table 5.25, Section 5.3.

5.2 evaluation mar2 and maps

5.2.1 *Experimental design*

Experiments are conducted to draw conclusions about the impact of different *factors* on sets of drawn samples. A factor, according to the terminology of experimental design, represents a characteristic of the testing environment that is assumed to have an impact on the samples. In our case, a sample corresponds to the fitness value of a particular solution and a factor to a parameter. Rardin et al. [28] suggest different problem instances also to be considered as factors, as different instance characteristics may also influence the solutions' quality. In this case we deal with two dimensions of factors, i.e. instances and parameters. However, in our case we will only look at the parameter dimension.

For each factor, different *levels* are tested. A factor level corresponds to a concrete parameter setting or a concrete problem instance.

We ran experiments for each level of each factor against the same set of problem instances. This method is also known as *blocking*, as opposed to *randomisation* [22] . Lin and Rardin [20] give arguments in favor of blocking compared to randomisation with respect to analysis of algorithms.

5.2.2 *Instances*

For the parameter evaluation we selected a set of six different instances among those presented by Beer at al. [6], which are publicly available in [31]. 20 of them were retrieved from a real life application without known optimal solutions, and ten selected among 60 randomly generated instances with known optimal solutions. The set of instances is the same as the one used by the authors of [6]. Details regarding the random generation are provided online by the same authors [31] .

The input data $\mathcal{C}$ (constraints) and k (number of timeslots) is 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 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 number of shifts and the shift lengths. As shown in Section 4.2, the domain size $\mathcal{D}_S$ grows exponentially with respect to the shift size $|S|$, which results in a smaller search space for instances with smaller shifts.

Table 5.9 gives an overview on the available instances. We renamed them for better readability.

5.2.3 *Parameters*

We first determined by trial and error a set of parameters influencing the search process along with relevant levels. In the course of experimention additional levels, which seemed worth testing, were added. The relevant set of parameters and levels is different for each of the algorithms. However, the following parameters have been evaluated for both algorithms

$|\mathcal{I}|$ Population size

κ Number of individuals (MAR2) or memes (MAPS) performing a tournament in the selection process

μ Search intensity: Number of iterations the local search continues without finding improvements

(η_1, η_2, η_3): Probability for each neighbourhood to be selected in each local search iteration

5.2.4 *Testing environment*

The algorithms were implemented with Comet [34], which is an object-oriented programming language specifically designed for constraint-based local search. Comet was also used by the authors of [6], which eases the comparison of results. Since we were dealing with a stochastic algorithm, we executed ten runs for each experimental setting. 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.

We benchmarked our machine and the one used by Beer et al. [6] in order to adjust our timeout and retrieve comparable results. This was done using the benchmark program provided by the organisers of PATAT 2008 timetabling competition [27], which determined the time a timetabling algorithm was allowed to run for the timetabling competition. The result retrieved by the benchmark program for Beer et al.'s machine was 468 on average for 10 runs and for ours 396. Beer et al. executed their algorithm for 3600 seconds, consequently our timeout was set to 3046 seconds. We analyse the solutions our algorithms produce after this timeout under different parameter settings.

5.2.5 *Evaluation method*

Different statistical methods exist to investigate the impact of different parameter settings, analysis of variance (ANOVA) (e.g. in [22]) being a widely used parametric method among them. This method assumes normal distribution in each of the groups, which for our case is difficult to test since our groups consist of only ten samples. We thus decided to use the Kruskal-Wallis test as non-parametric alternative to ANOVA. This method is also described in [22].

The Kruskal-Wallis method tests the null hypothesis that the samples of different groups were drawn from the same population [22] . This corresponds to the hypothesis that a parameter does not have a significant effect on the results. The null hypothesis is rejected if p is below an α-level usually chosen < 0.1.

The following tables show average results for each instance and parameter levels. The rightmost column indicates the level of significance obtained by applying the Kruskal-Wallis test on groups of ten runs for each level using the following legend:

******* standing for $p < 0.001$

****** standing for $p < 0.01$

***** standing for $p < 0.05$

. standing for $p < 0.1$

The lower p, the higher the significance of the impact of the parameter with respect to the levels tested. A blank field in the p column means that the null hypothesis could not be rejected.

Name	Alt. Name	Week	Forecast	# Shifts	# Slots
2fc04a	rl-1	2	A	135	15,656
2fc04a03	rl-2	2	A1	134	15,778
2fc04a04	rl-3	2	A2	137	15,870
2fc04b	rl-4	2	B	126	14,784
3fc04a	rl-5	3	A	124	14,740
3fc04a03	rl-6	3	A1	123	14,840
3fc04a04	rl-7	3	A2	128	14,980
3si2ji2	rl-8	3	C	151	16,324
4fc04a	rl-9	4	A	124	14,734
4fc04a03	rl-10	4	A1	123	14,828
4fc04a04	rl-11	4	A2	127	14,940
4fc04b	rl-12	4	B	125	14,564
50fc04a	rl-13	50	A	130	15,246
50fc04a03	rl-14	50	A1	130	15,372
50fc04a04	rl-15	50	A2	131	15,452
50fc04b	rl-16	50	B	126	14,952
51fc04a	rl-17	51	A	129	15,106
51fc04a03	rl-18	51	A1	129	15,226
51fc04a04	rl-19	51	A2	130	15,312
51fc04b	rl-20	51	B	126	14,830
random1-1	ran1-1	n/a	n/a	137	13,188
random1-2	ran1-2	n/a	n/a	164	15,144
random1-5	ran1-5	n/a	n/a	151	14,208
random1-7	ran1-7	n/a	n/a	137	12,780
random1-9	ran1-9	n/a	n/a	151	14,208
random1-13	ran1-13	n/a	n/a	124	12,288
random1-24	ran1-24	n/a	n/a	137	12,780
random1-28	ran1-28	n/a	n/a	124	11,412
random2-1	ran2-1	n/a	n/a	179	17,208
random2-4	ran2-4	n/a	n/a	162	15,444

Table 5.9: Overview on sample instances.

5.2.6 *Evaluation of MAR2*

We tested the following parameters for this algorithm:

$|\mathcal{I}|$ Population size

γ Crossover: Probability to select the fitter meme

α Crossover vs Mutation: Probability to create offspring by crossover, $1-\alpha$ probability to create offspring by mutation

κ Selection pressure: Number of individuals performing a tournament

λ Search rate, percentage of population the local search is applied on

$|L|$ Length of tabu list

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

(η_1, η_2, η_3): Probability for each neighbourhood to be selected in each local search iteration

Th population size made a significant difference in most of the instances, with the value performing best being $|\mathcal{I}| = 4$. We also tested a population size $|\mathcal{I}| = 1$ to verify if the genetic operators are relevant to the algorithm.

Interestingly, two of the parameters that were supposed to influence the genetic operators, i.e. γ and α did not have a significant impact. The parameter κ defining the selection pressure did have an impact on the solution qualities, but interestingly, 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 tabu search, i.e. λ, $|L|$, μ and η did have a significant impact 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, best results were obtained with a tabu list length of 0. The following tables summarise the average results over ten runs for each parameter and its values.

Inst. \ $\lvert\mathcal{I}\rvert$	1	10	20	4	40	70	p
random1-7	1090	1114	1120	1106	**1061**	1085	
rl-2	**3375**	3425	3426	3378	3426	3436	
rl-3	3316	3370	3398	**3301**	3381	3427	**
rl-4	2364	2420	2499	**2318**	2585	2731	***
rl-5	**1995**	2128	2089	2004	2212	2282	***
rl-6	1983	2059	2030	**1927**	2143	2200	***

Table 5.10: MAR2: Population size

Inst. \ γ	0.0	0.6	0.9	p
random1-7	1139	1120	**1100**	
rl-2	3431	3426	**3406**	
rl-3	**3371**	3398	3428	
rl-4	2507	2499	**2478**	
rl-5	2111	2089	**2070**	
rl-6	2062	2030	**2016**	

Table 5.11: MAR2: Crossover, preference for fitter memes

Inst. \ α	0.5	0.7	0.9	p
random1-7	1084	1061	**1057**	
rl-2	3435	3426	**3425**	
rl-3	3367	3381	**3358**	
rl-4	2656	**2585**	2639	
rl-5	**2157**	2212	2176	
rl-6	**2130**	2143	2157	

Table 5.12: MAR2: Crossover vs. Mutation

Inst. \ κ	1	2	3	p
random1-7	**1023**	1120	1105	**
rl-2	**3315**	3426	3474	**
rl-3	**3296**	3398	3411	*
rl-4	**2353**	2499	2451	**
rl-5	**1963**	2089	2108	**
rl-6	**1935**	2030	2061	**

Table 5.13: MAR2: Selection pressure

Inst. \ λ	0.1	0.2	0.5	0.8	p
random1-7	1012	920	894	**826**	***
rl-2	3324	3346	**3279**	3324	
rl-3	3269	3244	**3235**	3283	
rl-4	**2434**	2449	2468	2454	
rl-5	2028	2010	2002	**1991**	
rl-6	2005	1958	2014	**1938**	

Table 5.14: MAR2: Search rate λ

Inst. \ $\lvert L\rvert$	0	1	2	4	p
random1-7	**767**	772	842	885	**
rl-2	3275	**3217**	3241	3314	*
rl-3	**3193**	3213	3211	3240	
rl-4	2328	**2315**	2325	2440	**
rl-5	**1867**	1878	1892	2011	**
rl-6	**1806**	1835	1866	1922	*

Table 5.15: MAR2: Length of tabu list $|L|$

Inst. \ μ	10	15	2	6	8	p
random1-7	**854**	1010	999	938	885	***
rl-2	3357	3339	3572	3367	**3314**	***
rl-3	3306	3339	3459	**3219**	3240	***
rl-4	2428	**2327**	2831	2476	2440	***
rl-5	**1945**	1978	2367	2062	2011	***
rl-6	**1897**	1929	2298	2034	1922	***

Table 5.16: MAR2: Search intensity μ

Inst. \ η	1	2	nh1	nh2	nh3	p
random1-7	1624	**1131**	1446	1446	1601	***
rl-2	4201	3470	3486	**3411**	3809	***
rl-3	4014	3484	3412	**3395**	3706	***
rl-4	3600	2771	2734	**2658**	3161	***
rl-5	3158	2302	2294	**2247**	2688	***
rl-6	3134	2296	2250	**2148**	2597	***

Table 5.17: MAR2: Neighbourhoods η

The final runs for all real-life and randomly created instances were executed with the following parameters: $|\mathcal{I}| = 4$, $\gamma = 0.9$, $\kappa = 1$, $\alpha = 0.9$, $\lambda = 0.8$, $|L| = 0$, $\mu = 10$ and $\eta = (0.3, 0.6, 0.1)$. The results of those runs are compared to results from our other algorithms and results from literature in Section 5.3.

5.2.7 *Evaluation MAPS*

The following parameters were evaluated for this algorithm:

$|\mathcal{I}|$ Population size

λ Defines number of memes $|\mathcal{M}'|$ being mutated and improved for each individual: $\max(1, |\mathcal{M}| \cdot \lambda)$, $0 \leq \sigma \leq 1$

σ Mutation weight, number of shifts being mutated: $\max(1, |S'| \cdot \sigma)$, $0 \leq \sigma \leq 1$

κ Selection: Number of memes performing a tournament in the interaction operator

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

(η_1, η_2, η_3): Probability for each neighbourhood to be selected in each local search iteration

This 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. With $|\mathcal{I}| = 1$ no selection and no interaction is performed and thus the algorithm is reduced to a local search with mutation acting as perturbation. Since mutation may worsen a solution during the progress of the algorithm, for $|\mathcal{I}| = 1$ the best obtained solution is kept in memory. The results in Table 5.18 show clearly that there is indeed the need for a population based approach, as the results with runs applying local search only, i.e. with $|\mathcal{I}| = 1$, show the worst results.

The mutation and search rate λ determining $|\mathcal{M}'|$, the number of memes being improved on each individual led to the best results when kept low. On many instances, $\lambda = 0.05$ leads to only one memes being mutated and searched.

The mutation weight σ also worked well with a low value. σ determines the percentage of shifts which are assigned a new break pattern during a mutation.

Similar to the previous algorithms, the value of κ did not have a major impact.

As in MAR2, the local search intensity μ was set relative to the number of breaks $|B|$ taking part in the search. For this algorithm, larger values for μ probably performed better than for MAR2, because $|B|$ is much smaller. While in MAR2, B contained the complete set of break in the solution, MAPS 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.

We tested some more neighbourhood combinations than for MAR2. In Table 5.23 it can be seen very clearly, that 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 $\eta_1 = 0.8$ and $\eta_2 = 0.2$, that is, $\mathcal{N}_1$ chosen with a probability of 80% and $\mathcal{N}_2$ with a probability of 20%.

Instance \ $\lvert\mathcal{I}\rvert$	1	10	20	4	6	p
random1-7	1631	**667**	674	715	669	***
rl-1	3427	2969	3133	**2957**	2967	***
rl-16	2530	2070	2237	2012	**2008**	***
rl-19	2558	2120	2280	**2068**	2084	***
rl-4	2450	2018	2114	1966	**1957**	***
rl-7	2105	1786	1906	**1727**	1736	***

Table 5.18: MAPS: Population Size

Instance \ λ	0.05	0.1	0.2	0.3	0.5	p
random1-7	695	**694**	702	711	754	
rl-1	**2982**	2992	3033	3121	3223	***
rl-16	**1950**	2027	2048	2217	2263	***
rl-19	**2098**	2144	2155	2196	2311	***
rl-4	1986	**1985**	2038	2096	2165	***
rl-7	**1750**	1779	1819	1830	1929	***

Table 5.19: MAPS: Mutation and search rate

Instance \ σ	0.01	0.05	0.1	0.3	0.5	p
random1-7	**617**	624	674	667	657	.
rl-1	2947	2910	2924	**2905**	2971	
rl-16	1915	**1886**	1961	1974	1999	**
rl-19	2041	**1965**	2077	2022	2043	*
rl-4	**1904**	1905	1953	1949	1975	**
rl-7	1688	**1670**	1708	1702	1730	

Table 5.20: MAPS: Mutation weight

Inst. \ κ	1	2	p
random1-7	**676**	715	
rl-1	2975	**2957**	
rl-16	**1972**	2012	
rl-19	**2019**	2068	*
rl-4	1974	**1966**	
rl-7	1737	**1727**	

Table 5.21: MAPS: Selection κ

Instance \ μ	10	20	30	40	p
random1-7	669	618	630	**613**	.
rl-1	2987	**2910**	2937	2921	*
rl-16	2003	**1891**	1917	1927	*
rl-19	2056	2016	2026	**2004**	
rl-4	1968	1933	1916	**1911**	.
rl-7	1732	**1677**	1699	1699	

Table 5.22: MAPS: Search intensity

Inst. \ η	$(1,0,0)$	$(.2,.8,0)$	$(.5,.5,0)$	$(.8,.2,0)$	$(.3,.3,.3)$	$(.5,0,.5)$	$(0,1,0)$	$(0,.5,.5)$	p
random1-7	850	710	698	**640**	2027	2404	741	2059	***
rl-1	3427	2992	2975	**2931**	4334	4467	3070	4463	***
rl-16	2635	2111	2043	**1956**	4297	4528	2243	4323	***
rl-19	2634	2156	2099	**2070**	4105	4252	2212	4059	***
rl-4	2406	2032	2002	**1951**	3708	3951	2052	3833	***
rl-7	2118	1777	**1705**	**1705**	3204	3356	1810	3290	***

Table 5.23: MAPS: Neighbourhood

For the final runs we used the following settings: $|\mathcal{I}| = 4$, $\lambda = 0.05$, $\sigma = 0.05$, $\mu = 20$, $\kappa = 1$ and $\eta = (0.8, 0.2, 0.0)$. The results of those runs are compared with results from our other algorithms and results from literature in Section 5.3.

5.3 COMPARISON OF RESULTS

The following tables compare the results of the final runs for MAR1, MAR2 and MAPS with the best values provided by literature [6]. Each column indicates the best and average results as well as the standard deviation σ over 10 runs. The timeout has been normalised as described in Section 5.2.4 to make the results comparable.

The table includes two result columns from [6]. Each represents values retrieved by a min-conflicts heuristic. They differ in the initialisation and the definition of hard- and soft constraints. While the values in the first column ("STP init") were retrieved by defining all constraints $\mathcal{C}$ as hard constraints, thus never allowing any violation and using the small temporal problem model [15], the values in the second column ("Random init") define $\mathcal{C}$ as soft constraints and resolve them along with optimising the staffing requirements. In the "Random init" version, violations in $\mathcal{C}$ are included in the objective function. According to the authors, over 20 real life instances no significant difference between the two approaches could be found.

Compared within our algorithms, MAR1 performs worst on most of the random instances, except for one. Since the main difference between MAR1 and MAR2 is the memetic representation, we conclude that Memetic Representation 2 indeed outperforms Memetic Representation 1. Additionally, the standard deviation σ is lower for the results retrieved with Memetic Representation 2, which makes MAR2 and MAPS more reliable methods.

The best results, however, were returned by MAPS. Using this algorithm, we managed to set new upper bounds for 18 out of 20 real life instances. Also on the random instances this algorithm returns better results than both the algorithms from literature and the algorithms from this work. Additionally, in most instances the solutions returned by MAPS have a lower standard deviation than any of the other algorithms.

Details on the solutions are presented in Table 5.24 showing best and average values as well as the standard deviation over ten runs for each real life instance for the algorithms presented by [6], MAR2 and MAPS. Solutions for random instances are presented in Table 5.25 comparing the algorithm presented by [6], MAR1, MAR2 and MAPS.

Instance	[6] STP init			[6] Random init			MAR2			MAPS		
	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ
2fc04a	3,094	3,248	84	3,112	3,224	86	3,244	3,326	50	**2,816**	**2,961**	71
2fc04a03	3,100	3,229	61	3,138	3,200	39	3,220	3,328	57	**2,834**	**2,934**	54
2fc04a04	3,232	3,371	68	3,234	3,342	60	3,226	3,297	44	**2,884**	**2,954**	60
2fc04b	2,017	2,104	92	**1,822**	2,043	99	2,266	2,387	68	1,884	**1,948**	49
3fc04a	1,746	1,809	49	1,644	1,767	102	1,810	1,909	59	**1,430**	**1,533**	67
3fc04a03	1,632	1,804	87	1,670	1,759	53	1,846	1,944	55	**1,440**	**1,514**	40
3fc04a04	1,942	2,032	51	1,932	1,980	40	1,930	2,056	87	**1,614**	**1,718**	48
3si2ji2	3,626	3,692	35	3,646	3,667	14	3,344	3,398	27	**3,177**	**3,206**	17
4fc04a	1,694	1,851	126	1,730	1,817	48	1,814	1,972	139	**1,478**	**1,540**	29
4fc04a03	1,666	1,795	87	1,748	1,834	55	1,742	1,870	59	**1,430**	**1,502**	42
4fc04a04	1,918	2,017	95	1,982	2,064	62	1,850	1,980	60	**1,606**	**1,674**	48
4fc04b	1,440	1,527	56	1,410	1,489	49	1,628	1,772	154	**1,162**	**1,233**	48
50fc04a	1,750	1,861	95	1,672	1,827	81	2,018	2,090	32	**1,548**	**1,603**	36
50fc04a03	1,718	1,847	96	1,686	1,813	84	1,822	1,951	87	**1,402**	**1,514**	67
50fc04a04	1,790	1,985	83	1,792	1,917	64	1,914	2,009	48	**1,480**	**1,623**	89
50fc04b	1,854	2,012	91	1,822	1,954	77	2,322	2,464	98	**1,818**	**1,900**	56
51fc04a	2,048	2,204	89	2,054	2,166	62	2,490	2,836	687	**1,886**	**2,074**	87
51fc04a03	2,004	2,096	60	1,950	2,050	86	2,318	2,377	37	**1,886**	**1,949**	46
51fc04a04	2,058	2,195	64	2,116	2,191	53	2,370	2,728	678	**1,958**	**2,039**	52
51fc04b	2,380	2,514	106	**2,244**	2,389	94	2,796	2,950	88	2,306	**2,367**	43

Table 5.24: Comparison with literature: Real life instances

Instance	[6] STP init			MAR1			MAR2			MAPS		
	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ
random1-1	728	972	177	672	738	56	544	592	41	**346**	**440**	48
random1-2	1,654	1,994	172	2,174	2,352	89	712	817	92	**370**	**476**	65
random1-5	1,284	1,477	99	870	1,044	86	696	742	47	**378**	**418**	29
random1-7	860	1,077	154	742	861	68	824	940	73	**496**	**583**	42
random1-9	1,358	1,658	213	1,918	2,095	108	672	734	38	**318**	**423**	51
random1-13	1,264	1,535	245	2,884	3,039	98	570	699	68	**370**	**445**	55
random1-24	1,586	1,713	74	1,182	1,330	107	884	934	46	**542**	**611**	43
random1-28	1,710	2,020	233	2,926	3,018	63	626	726	71	**222**	**318**	71
random2-1	1,686	1,855	142	1,262	1,537	117	914	1,058	91	**724**	**889**	75
random2-4	1,712	2,053	242	2,182	2,336	111	794	889	56	**476**	**535**	45

Table 5.25: Comparison with literature: Random instances

6 CONCLUSIONS AND FUTURE WORK

In this thesis we proposed different memetic approaches to optimise Bsp. We introduced two different representations upon which we built three different memetic algorithms. For each method we designed a set of genetic operators. As local improvement we applied in all approaches the same local search heuristic, for which we proposed three different neighbourhoods. Further, we introduced a concept to avoid local optima based on penalty values for parts of solutions which are not improved during too many iterations.

We justified the choice of a metaheuristic to optimise Bsp by presenting an NP-completeness proof for this problem under the condition that the input contains break patterns explicitly instead of defining them by a set of constraints.

For each algorithm we conducted a set of experiments on different parameter settings and then compared the algorithms with their best settings. The impact of each parameter was assessed with statistical methods. Important outcomes of these parameter evaluations are the following:

- Using genetic operators combined with local search returns better results than using only local search.
- Applying the local search either only on some individuals or only on small parts of each individual significantly improves the qualities of the solutions compared to applying the local search on all and entire individuals.
- Focusing on neighbourhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ returns better solutions than using only one neighbourhood. The largest neighbourhood performs worst, the smallest best.
- 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 improves the qualities of the solutions.

The results of the algorithm performing best according to our experiments were compared to the results in literature according to 30 publicly available benchmarks.

Our algorithm returned improved results for 28 out of 30 instances. To the best of our knowledge, our results are the new upper bounds for the improved instances.

We leave the computational complexity of Bsp with break patterns implicitly given by a set of constraints as open issue. As mentioned in Section 2.3, to the best of our knowledge, it is still unclear whether solving a shift scheduling problem with a large number of breaks or approaching the problem into two phases, namely a shift scheduling and a break scheduling phase, is more effective in practice. Future work could include an investigation on the performance of metaheuristics on these two different optimisation approaches.

Another interesting topic that was not addressed in this work is the question on how the algorithms perform in very long runs, e.g. up to ten hours. The execution of the algorithm could also be accelerated by parallelization.

BIBLIOGRAPHY

[1] S. Atran. *In Gods We Trust: The Evolutionary Landscape of Religion*. Oxford University Press, 2002. ISBN 978-0195149302.

[2] T. Aykin. A comparative evaluation of modelling approaches to the labour shift scheduling problem. *European Journal of Operational Research*, 125:381–397, 2000.

[3] T. Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42:591–603, 1996.

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

[5] A. Beer, J. Gaertner, N. Musliu, W. Schafhauser, and W. Slany. An iterated local search algorithm for a real-life break scheduling problem. In *Matheuristics 2008- Second International Workshop on Model Based Metaheuristics*, Bertinoro, Italy, 2008.

[6] A. Beer, J. Gaertner, N. Musliu, W. Schafhauser, and W. Slany. A break scheduling system using AI techniques. *IEEE Intelligent Systems*, 2008.

[7] S.J. Blackmore. *The meme machine*. Oxford University Press, 2000.

[8] A. Brindle. *Genetic algorithms for function optimisation*. PhD thesis, University of Alberta, Department of Computer Science, Edmonton, Canada, 1981.

[9] E.K. Burke, D.G. Elliman, and R.F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *6th International Conference on Genetic Algorithms*, pages 1258–1271, Pittsburgh, USA, 1995.

[10] E.K. Burke, J.P. Newall, and R.F. Weare. *Handbook of Combinatorial Optimization*, volume 3, chapter Tabu Search, pages 621–757. Kluwer Academic Publishers, 1999.

[11] C. Canon. Personnel scheduling in the call center industry. *4OR: A Quarterly Journal of Operations Research*, 5(5(1)):89–92, 1989.

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

[13] C.R. Darwin. *The origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. London: John Murray, 1859.

[14] R. Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

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

[16] D.C. Dennett. *Consciousness Explained*. Little, Brown and Co., 1991. ISBN 0316180653.

[17] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[18] F. Glover and M. Laguna. *Lecture Notes in Computer Science*, chapter A Memetic Algorithm for University Exam Timetabling, pages 241–250. Springer Berlin/Heidelberg, 1996.

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

[20] B.W. Lin and R.L. Rardin. Controlled experimental design for statistical comparison of integer programming algorithms. *Management Science*, 25:1258–1271, 1980.

[21] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4:337–351, 2000.

[22] D.C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 6 edition, 2005.

[23] 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.

[24] P. Moscato and M.G. Norman. *Parallel Computing and Transputer Applications*, chapter A Memetic Approach for the Traveling Salesman Problem – Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems, pages 177–186. IOS Press, 1992.

[25] Österreichisches Arbeitszeitgesetz, paragraph 11, 2009. URL http://www.ris.bka.gv.at/Dokumente/Bundesnormen/NOR12113700/NOR12113700.pdf.

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

[27] PATAT Benchmark program, 2008. URL http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm.

[28] R.L. Rardin and R. Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7:261–304, 2001.

[29] M. Rekik, J.F. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13:49–75, 2010.

[30] M. Segal. The operator-scheduling problem: A network flow approach. *Operations Research*, 22:808–823, 1974.

[31] Shift Design and Break Scheduling Benchmarks, 2008. URL http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/.

[32] P. Tellier and G. White. Generating personnel schedules in an industrial setting using a tabu search algorithm. In H. Rudova E. K. Burke, editor, *PATAT 2006*, pages 293–302, 2006.

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

[34] P. Van Hentenryck and L. Michel. *Constraint-based local search*. Massachusetts Institute of Technology, 2005.

[35] T. Yamada, K. Yoshimura, and R. Nakano. *Simulated Evolution and Learning*, chapter Information Operator Scheduling by Genetic Algorithms, pages 50–57. Springer Berlin / Heidelberg, 1999.