Let the weekend begin!

A solution for solving the Weekend Scheduling Problem for ORTEC Harmony

Master thesis Industrial Engineering and Management

Frédérique Versteegh
frederiqueversteegh@gmail.com

June, 2009

Committee:
Dr. ir. E.W. Hans (University of Twente)
Dr. ir. G.F. Post (University of Twente)
Drs. M. Hoogstrate (ORTEC)

University of Twente
School of Management and Governance
Department of Operational Methods for Production and Logistics

Preface

Starting to write your thesis feels like finishing a great period of life. Finishing writing your thesis feels like starting a new period in life. I thank all my friends and my family for the great experience that my student time was.

Graduation is sometimes a rough experience, but every rough moment forces you to learn and to grow. I thank ORTEC for offering me this research project and especially for the chance they have given me to go to Calgary. This was a very challenging and interesting experience in which ORTEC gave me the confidence of successfully accomplishing the project. I thank my supervisor at ORTEC, Monique Hoogstrate, for her support, her critical views, the valuable lunch breaks, and for the nice period we have had. I thank my colleagues at ORTEC for their support, and especially my roommate, Egbert van der Veen, who I have bothered with many questions.

I thank my first supervisor at the University of Twente, Erwin Hans, for his enthusiasm, his critical views, and for challenging me to a higher level. I thank Gerhard Post for his role as second supervisor.

Graduation knows some tough periods. I thank my boyfriend, Ralf Stamps, and my two dear friends, Golein Klein Bramel and Els Hettinga, for brightening me up in hard times and for not blaming me that I was not always in the best mood.

Last but not least, I thank my parents and my brothers for their support, the chances they have given me, and the freedom to explore.

To learn is to discover things of which you did not even know that you did not know them.

Management summary

Introduction

This research is performed within the Product Knowledge Center of ORTEC. ORTEC is one of the largest providers of advanced planning and optimization software solutions and consulting services. We study the assignment of weekend shifts in ORTEC's decision support software solution for workforce scheduling, called ORTEC Harmony. The assignment of weekend shifts is of great importance to customers, as weekends have an impact on the social life of employees and thus on employee satisfaction. In addition, unfulfilled demand in weekends is hard to cover and expensive, as irregularity allowances have to be paid for those hours. Harmony has the functionality to automatically create schedules. In a case study we analyze three real-life cases from practice that use this functionality. These customers use a two-step approach to schedule: weekend shifts as a priori step, followed by the remaining shifts. The current alternatives in Harmony to create satisfying schedules are very time-consuming. Even the best approach (varies per user) regularly leads to unfulfilled weekend demand and does not always satisfy user requirements. Users prefer an equitable schedule in which every employee works approximately the same number of weekend shifts and shift types.

Research objective

The objective of this research is to improve the assignment of weekend shifts such that unfulfilled demand is minimized and user requirements are satisfied as much as possible. The problem addressed by this objective is referred to as the Weekend Scheduling Problem.

Solution approach

We model the Weekend Scheduling Problem as a Quadratic Integer Programming model. For this problem we propose a stand-alone solution, the Weekend Planner. The Weekend Planner can be

embedded in Harmony, both as an individual planner and as an element of the current optimization engine. The Weekend Planner assigns weekend shifts to employees using an intelligent form of list scheduling. Working a shift means working a specific shift type in a specific weekend, for the whole weekend. For the selection of shifts and resources, the Weekend Planner uses selection rules based on flexibility and busyness. First, the least flexible shift is selected. In case of a tie, the shift is selected based on randomness. Second, the least busy and least flexible resource is selected. Again, in case of a tie, the resource is selected based on randomness. Constraints on availability and on the maximum allowed number of weekends to work are taken into account. The creation of one schedule ends with a local search method. Random sampling is used to create diversity in the resulting schedules.

Results

The Weekend Planner is tested on three real-life cases from practice and on 400 random instances. Randomly generated instances with parameter ranges typically seen in practice are used to test the robustness and sensitivity of the Weekend Planner. Computational experiments show that the Weekend Planner solves 89% of the instances - i.e., all shifts are assigned - and solves 67.5% of the instances to optimality. The results deviate on average 3.8% from optimal. The Weekend Planner is most sensitive for cyclical schedules and schedules in which every employee needs to work its maximum number of weekends, in other words a tight schedule.

Computational experiments on the three real-life cases show that the Weekend Planner outperforms Harmony on all aspects and improves schedules by 15% to 400%.

Conclusions

The Weekend Planner is a suitable method for solving the Weekend Scheduling Problem. The Weekend Planner decreases the unfulfilled demand and delivers more equitable schedules than Harmony does. The Weekend Planner focuses on weekend related constraints and therefore is a user-friendly and not time-consuming method.

Recommendations

We recommend to use a more extensive local search method in the Weekend Planner, which is already available in Harmony, to decrease the deviation from optimal and decrease the number of unassigned shifts, if any. Finally, we recommend to implement the Weekend Planner in Harmony.

Contents

Chapter 1

Introduction

This research describes improvements for a workforce scheduling algorithm that is used in a software solution for workforce scheduling called *ORTEC Harmony* (in the remainder called Harmony), a product of ORTEC. ORTEC is one of the largest providers of advanced planning and optimization software solutions and consulting services.

This research analyzes a problem in one of Harmony's modules: the optimization engine, which is used to automatically generate workforce schedules. First, it analyzes the current performance of Harmony regarding the planning of weekend shifts using a case study. The main current issues are inability to cover demand and to create an equitable division of weekend shifts. To solve these issues we propose a heuristic approach, the Weekend Planner, for the assignment of weekend shifts. The Weekend Planner consists of an intelligent form of list scheduling followed by a local search. The selection rules used in the list scheduling are based on selecting the least flexible shift and the least flexible resource. We describe the implementation of the Weekend Planner and test its robustness and the sensitivity of the solution. Finally, we compare the results of the Weekend Planner with Harmony's results for three real-life cases from practice.

Section 1.1 shortly describes the context in which this research has taken place and discusses ORTEC and Harmony. Section 1.2 describes the problem, followed by the research objectives and research questions in Section 1.3.

1.1 Context

ORTEC

ORTEC is one of the largest providers of advanced planning and optimization software solutions and consulting services. ORTEC's solutions result in optimized fleet routing and dispatch, vehicle and pallet loading, workforce scheduling, delivery forecasting and network planning. ORTEC has over 700 employees in several offices in Europe and North America and an extensive customer base in a large number of industries, including:

- trade, transport, and logistics
- oil, gas, and chemicals
- consumer packaged goods
- health care
- professional and public services

This research takes place at ORTEC's Product Knowledge Center (PKC) in Gouda, the Netherlands. PKC transfers product knowledge and delivers finished products to market units and partners and is the link between ORTEC Software Development (OSD) and the market units. They are responsible for release management, documentation and trainings, product consultancy, and support. OSD also supports this research. OSD is responsible for developing and building ORTEC software.

Harmony

Harmony is an advanced planning solution for workforce scheduling. It is used in organizations that operate in an environment where work is carried out at irregular times and/or where the workload is fluctuating during operating hours. Typical customers can be found in health care and in security industry. Harmony offers the possibility of creating shift rosters in a rapid and well-organized manner. One of its goals is to automatically generate the closest to optimal workforce schedules for a set of resources (employees) and a set of shifts, constrained by rules (hard constraints) and criteria (soft constraints).

Harmony provides the functionality to create a schedule in two ways: manually, by letting the user assign shifts to employees, or automatically, by using the optimization engine that is included

in the software. In both ways the user is able to define legislation, company specific rules, and regulations in the software. Harmony takes these into account to check if it is allowed to assign a shift to an employee on a certain day.

Harmony has two different types of schedules: cyclical schedules and non-cyclical schedules. Cyclical schedules are schedules repeating at regular intervals (i.e. every 4 weeks). A non-cyclical schedule is a schedule with a start and end date.

1.2 Problem description

The performance of the optimization engine does not meet several important requirements of some customers. In a case study (Section 2.2) we describe three customer situations, both cyclical and non-cyclical, and the problems they encounter when using the optimization engine. In that case study, the optimization engine appears not to be able to deliver schedules that meet or improve the quality of manually created schedules. ORTEC feels the need to investigate the abilities to improve the performance of the optimization engine. This results in the following problem description:

The optimization engine in Harmony does not meet all requirements regarding the assignment of shifts.

1.3 Research objective

In a preliminary study (Appendix A), we found that having a good assignment of weekend shifts is crucial for creating a good schedule. In that same study we found that one of the main performance problems of the optimization engine for cyclical schedules is its inability to create a proper assignment of weekend shifts. The following case study (Section 2.2) examines two more cases and shows that this problem not only arises in the optimization engine for cyclical schedules, but also in the optimization engine for non-cyclical schedules. This makes the inability to create a proper assignment of weekend shifts a generic problem within Harmony. To improve the overall performance of Harmony we thus have to improve the assignment of weekend shifts. The objective of this research is to develop a solution with a performance improvement that results in the same number of unassigned weekend shifts (or less) and an improved score on key performance indicators as defined by users in the case study. We call this solution the Weekend Planner. We expect that by minimizing the number of unassigned weekend shifts, the total number of unassigned shifts is minimal as well, as the weekend shifts are the most crucial shifts

in the schedule. The general principle of Harmony is that the first priority is to assign as many shifts as possible, while minimizing the penalties for soft constraints. Therefore, penalties on non-key performance indicators are of less importance in this research. This results in the following research objective:

Improve the assignment of weekend shifts for ORTEC Harmony, such that the total number of unassigned shifts and the penalties on key performance indicators are reduced.

To be able to achieve the research objective, this research investigates the following subjects:

- Context
  1. *How does the algorithm currently work in Harmony?*
     Description of the algorithm and current possibilities to plan in Harmony (Section 2.1).
  2. *What are the current problems with weekend shifts in more detail?*
     Analysis of three real-life cases from practice, performance, and scheduling alternatives (Sections 2.2 - 2.4).
- Model
  1. *What is the mathematical formulation of the problem?*
     Definition of the Weekend Scheduling Problem and a Quadratic Integer Programming formulation (Sections 3.1 - 3.3).
- Algorithmic improvements
  1. *What requirements should be considered when deciding on the algorithmic improvement?*
     Summarize found requirements in previous chapters (Section 4.2).
  2. *How can we solve the Weekend Scheduling Problem?*
     Description and design of the Weekend Planner (Chapter 4).
- Results
  1. *Is the solution robust?*
     Approach of experiments and results of tests (Sections 5.1- 5.3).

2. *How sensitive is the solution for various parameters?*

   Sensitivity analysis for algorithm settings and instance parameters (Section 5.4).

3. *What is the performance result of implementing the improved algorithm, when testing on the case study?*

   Analysis of Harmony's results compared with results of the Weekend Planner (Section 5.6).

In this report we will use performance and implementation characteristics as measures for the algorithm. By performance we mean the quality of the schedule, which is measured in number of unassigned shifts and the amount of penalties for not meeting soft constraints. Implementation characteristics are for example the flexibility of the algorithm, the running time, independence of third-party software, and complexity of the implementation.

Chapter 2

Context

This chapter discusses the context and some background information on this research. Section 2.1 explains how the optimization engine in Harmony currently works, including explanation of constraints, the different automatic planners available in Harmony, a model formulation, and the algorithm itself. Section 2.2 discusses a case study in which we analyze the problems regarding weekend shifts of three users of Harmony. Section 2.3 summarizes the current performance of the optimization engine and Section 2.4 describes the requirements for the solution. For information used in this chapter we refer to Fijn van Draat et al. (2005) and Van der Put (2005).

2.1 Harmony's optimization engine

The goal of the optimization engine of Harmony is to generate qualitatively good schedules, such that for as many shifts as possible the required number of employees is assigned, while all hard constraints and as many soft constraints as possible are satisfied. Hard constraints make schedules that do not satisfy these constraints infeasible; soft constraints act as quality measures for a schedule. Section 2.1.1 discusses the hard and soft constraints used in Harmony. Section 2.1.2 discusses the mathematical formulation of the workforce scheduling problem that Harmony solves. Section 2.1.3 explains the different automatic planners available in Harmony. Section 2.1.4 describes the algorithm of the optimization engine.

2.1.1 Hard and soft constraints

Numerous constraints are involved in personnel scheduling, both hard and soft constraints. Hard constraints impose rules on the schedule, making it infeasible when one of the hard constraints is violated. The optimization engine in Harmony never violates a hard constraint. The soft constraints are a measure for the quality of a schedule. Not satisfying a soft constraint increases the cost of the schedule, or in other words: imposes a penalty on the schedule (see formula (2.2) in Section (2.1.2)). In Harmony, the hard constraints are mostly incorporated in the so-called *labor rules*; the soft constraints are incorporated in the *scheduling criteria* (valid for all employees in the schedule) and *work agreements* (apply to individual employees).

Hard constraints

The hard constraints can be divided into four categories:

1. *Legislation*: these constraints apply to all employees. Examples of these constraints are:
   - A maximum of x hours of work in y week(s)
   - In x days there should be a period of rest of at least y hours
   - After a night shift the rest time should be at least x hours
2. *Industry agreements*, such as collective labor agreements: these constraints apply to all employees that work in the sector or industry for which the agreement has been set. Examples of these constraints are:
   - At least x weekends off in a period of y weeks
   - A weekend off is defined as a time slot of at least x hours without labor, including Saturday 00:00 AM until Monday 04:00 AM.
3. *Organizational/departmental level*: extra constraints can be defined within an organization or a department. An example of such a constraint is:
   - At most x shifts of work in a row
4. *Personal level*: constraints that only apply to certain employees, based on their individual contract or requests. Examples of these constraints are:
   - An employee should (not) work on a specific day of the week

- An employee can only work shifts of type x and y
- An employee can only work shifts for which he is qualified

The constraints mentioned above are just examples for the specific categories, some constraints can appear in more categories with different parameter values. For example, legislation can subscribe a maximum number of 60 hours per week, but for the specific industry rules may say that a lower maximum number of 50 hours per week applies.

Soft constraints

The soft constraints can be divided into the following three categories:

1. *Organization or department criteria*: these apply to all employees of the organization/department. Examples of these criteria are:
   - No stand-alone shifts
   - In a weekend assign either no, or otherwise x shifts
2. *Group criteria*: apply to certain groups of employees, such as part-time employees. Examples of these criteria are:
   - Shifts series should be between x and y shifts for employees working v to z hours per week
   - At least x shifts of type y in a period of w weeks for employees working v to z hours per week
3. *Individual criteria*: preferences of individual employees, such as:
   - Employee x prefers shift y on day z

2.1.2 Model formulation

In Harmony, a penalty or cost function is used to define the quality of a schedule. The cost of the schedule increases with a penalty each time a criterion is violated. To be able to distinguish between important and less important soft constraints, each constraint has a weight and a severity type assigned to it. The user sets the weight for each criterion, which is multiplied with the severity when a criterion is not met. The severity depends on the excess of the value of the

criterion (how much over or under the defined criterion boundaries) and on the way the severity is calculated (depending on the criterion the excess is taken linearly or quadratically). The severity thus differs every time a criterion is broken. An example of severity is when the criterion describes a maximum of 4 shifts in a row and 6 shifts in a row are planned, then the criterion is exceeded by 2 shifts. If the excess of this criterion is taken quadratically, the severity is 4.

The value of the cost function is the weighted sum of the unsatisfied soft constraints. A qualitatively good schedule is a schedule with a minimum number of unassigned shifts and an objective value of the cost function that has a minimum amount of penalties given the number of unassigned shifts. We call a schedule an individual, the objective is to improve the score of that individual. The score of an individual consists of the penalties per resource and the penalties for the unassigned shifts. The weight used for the criterion that involves the unassigned shifts is usually set to 1,000,000, whereas weights for other criteria are in the range of 1 to 1,000. This section discusses the problem description of Harmony's workforce scheduling model. First we introduce some notation:

- R is the set of resources (employees), r is an element of the set if $r \in R$
- S is the set of shifts, s is an element of the set if $s \in S$
- C is the set of criteria, c is an element of the set if $c \in C$
- w_c is the weight assigned to criterion c
- w_{c_u} is the weight for an unassigned shift (= 1.000.000)
- ς_c is the severity of how criterion c is not met
- $\varphi(r,c) = \begin{cases} \varsigma_c & \text{if resource r violates criterion c} \\ 0 & \text{otherwise} \end{cases}$
- $\tilde{\varphi}(s) \quad = \begin{cases} 1 & \text{if shift s is unassigned} \\ 0 & \text{otherwise} \end{cases}$

The cost function per resource $f(r)$ can be defined as:

$$f(r) = \sum_{c \in C} w_c \cdot \varphi(r,c) \tag{2.1}$$

The score for an individual schedule consists of the score of all resources and the penalty for unassigned shifts: the higher the score is, the better. The score for an individual can then be

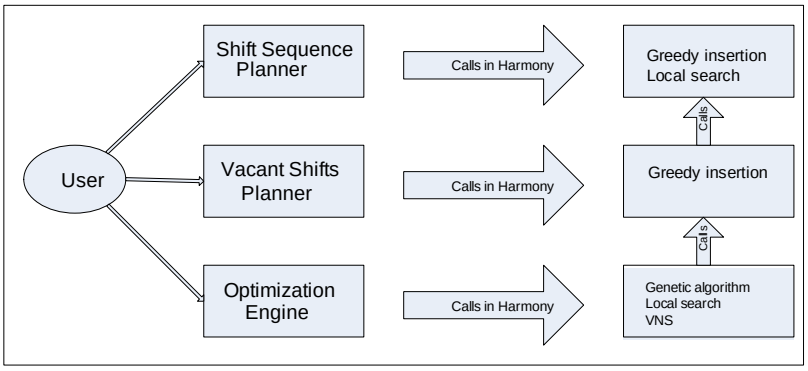

Figure 2.1: Relation between automatic planners in Harmony

defined as:

$$Score\ Individual = -\sum_{r \in R}\sum_{c \in C} w_c \cdot \varphi(r,c) \; - \sum_{s \in S} w_{c_u} \cdot \tilde{\varphi}(s) \tag{2.2}$$

2.1.3 Automatic planners

Harmony contains three different automatic planners:

- *Shift Sequence Planner*: the user defines possible shift sequences: a consecutive combination of shifts that occur often, i.e. DDDD (four day shifts in a row) or DDEE (two day shifts followed by two evening shifts). This planner tries to schedule as many shift sequences against the lowest possible cost, using a greedy insertion followed by a local search.

- *Vacant Shifts Planner*: assigns the unassigned shifts using a greedy insertion. This planner sorts the unassigned shifts based on several criteria and assigns each shift to the best available resource (based on cost function) for which the rules are not violated. This Vacant Shifts Planner does not reconsider assigned shifts.

- *Optimization engine*: optimization engine with a genetic algorithm and a local optimizer that generates a complete schedule. Section 2.1.4 describes the algorithm of the optimization engine in more detail.

The user can call these three planners individually. Internally however, there is interaction between these planners. Figure 2.1 shows the relation between the three described planners. Our research focuses on the optimization engine. In the next subsections we extensively discuss the algorithm currently used in the optimization engine.

2.1.4 Algorithm of Harmony's optimization engine

This section discusses the algorithm of Harmony's optimization engine's, which consists of three steps:

1. Genetic algorithm
2. Local optimization
3. Variable Neighborhood Search (VNS)

The philosophy behind this algorithm is twofold:

- *robust* - the algorithm is robust for instance characteristics (input data) and problem definitions (changing constraint and/or criteria set);
- *control possibilities* - the planner can adjust weights for criteria etc.

The remainder of this section describes the three steps of the algorithm.

Genetic algorithm

The first phase of the algorithm consists of a genetic algorithm. A genetic algorithm is a randomized search process based on the principles of natural evolution and 'survival of the fittest'. For general information on genetic algorithms we refer to Goldberg (1989). The first step of the algorithm is to generate an initial population, called *generation 0*. The user sets a value for the number of individuals in the population that has to be created. Harmony uses the Vacant Shifts Planner to create generation 0. To make sure to get different unique individuals, the algorithm divides the resources randomly in two groups and uses the Vacant Shifts Planner to first assign as many shifts as possible to the resources of the first group and then assigns as many remaining shifts as possible to the resources of the second group. This procedure is repeated until the algorithm has created enough unique individuals for generation 0.

The next step of the genetic algorithm consists of a global optimization until no further improvements can be found within an acceptable amount of time. The global optimization uses mutation- and crossover operators to create new individuals out of the existing population and uses selection methods to extract the next generation out of the current population. The 'fittest' individuals, the ones with the best score on the cost function, have a better chance to survive into the next generation, but it depends on the selection method which individuals make it to

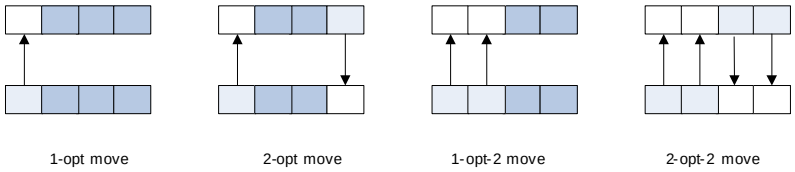

Figure 2.2: 1-opt and 2-opt moves

the next generation. This process of creating generations continues until no further improvement is found during a certain period.

Local optimization

Once the global optimization stops, the algorithm continues with a local search consisting of a 1-opt and 2-opt search on the best schedule found in the genetic algorithm. A 1-opt move (Figure 2.2) takes one shift and assigns it to the first feasible resource that is better than the current assignment. A 2-opt move consists of selecting two shifts that are currently assigned to different resources. If exchanging them is feasible and results in an improvement, the exchange takes place. The schedule is said to be 1-optimal when the schedule cannot be improved by a 1-opt change. When the schedule is 1-optimal the algorithm starts a 2-opt search. When it finds one 2-opt improvement, the algorithm returns to the 1-opt search, until the schedule is 1-optimal again. When the schedule is also 2-optimal, the algorithm starts a 1-opt-2 search: a 1-opt search in which 2 shifts are moved to another resource. The algorithm returns to the 1-opt-1 search every time an improvement is found, it continues the local search until it reaches a 2-opt-3 optimal schedule, or until no further improvements can be found within a reasonable amount of time.

Variable Neighborhood Search

When the algorithm reaches local optimality with respect to 1- and 2-opt moves, the third part of the optimization engine starts: Variable Neighborhood Search (VNS). The VNS tries to further improve the best solution so far by escaping from the found local optimum. VNS 'kicks' the schedule to another neighborhood and finds the local optimum in that neighborhood using the second phase of this algorithm (Figure 2.3). If this local optimum is better then the one found so far, the algorithm continues with this schedule. The 'kick' consists of either unassigning all shifts of three to five resources, which are drawn with a chance proportional to their scores, or unassigning a random block of 25 or 50 shifts. If a kick does not result in a schedule with an improved score, the algorithm goes back to the current best schedule and tries a new kick. The

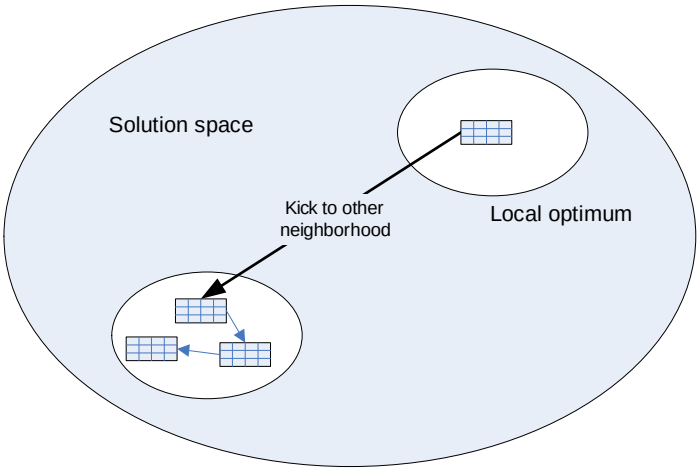

Figure 2.3: Variable Neighborhood Search

algorithm continues until the time set by the user is spent or until an optimal solution is found, in other words when a schedule is found with a value 0 for the objective function as described in formula 2.2. When the value of formula 2.2 is 0, the schedule has no unassigned shifts and violates no soft constraints.

2.2 Case study

Appendix A describes a preliminary study performed at a customer of Harmony: Calgary Health Region (CHR). CHR uses a new module in Harmony: the optimization engine for cyclical schedules. In this study the problem of assigning weekend shifts as described in Chapter 1 first became clear. This section summarizes their problems with respect to weekend shifts. To test whether this performance problem only appears in the new functionality, this section also discusses two cases of other ORTEC customers that use the optimization engine for non-cyclical schedules: the Belgian police and a Dutch hospital, the 'Kennemer Gasthuis Haarlem'.

Calgary Health Region

At Calgary Health Region various problems arise with respect to weekend shifts. As mentioned before, Calgary Health Region uses cyclical schedules. The most important problem they encounter is that the optimization engine is not able to assign all required weekend shifts to employees, while the resulting unassigned shifts can be assigned if the user makes a manual assignment. Next, Calgary Health Region wants weekend shifts to be assigned for a complete weekend, they

do not want employees to work only a Saturday or only a Sunday. In the current performance of Harmony single weekend shifts sometimes are assigned. A third problem is that the assignment of weekend shifts is not equitable, not regarding the amount of the same shift types and not regarding the number of weekends an employee works. This last problem arises especially when more employees than weekend shifts are available, in which case the weekend shifts are not divided equally over all employees. This results for example in one employee working three weekends and somebody else working only one weekend. It is neither equitable when one employee works two weekends of night shifts and another employee works two weekends of day shifts.

Belgian Police

The Belgian Police also uses the optimization engine in Harmony to schedule their employees. They use non-cyclical schedules. The problems they encounter when making schedules are comparable to those at Calgary Health Region: not all shifts are assigned to employees, not every employee works the same number of weekend shifts, and employees get assigned single weekend shifts. The first problem arises especially when employees already have for example a vacation, training, or occasional activities (such as alcoholic beverage control) scheduled in the scheduling period, so more constraints are placed on that resource. They also want an equitable division of the types of shifts to be worked among employees. Their workaround to create reasonable schedules is to use the Shift Sequence Planner and manual tweaking for the weekend shifts followed by the optimization engine to schedule the remaining shifts. Nevertheless, this workaround does not solve all problems mentioned before. They still encounter difficulties to divide weekend shifts equally over employees when they already have leave scheduled in the scheduling period. They use different rules than Calgary Health Region when scheduling for weekends; all rules that do not directly influence weekend shifts are turned off. They do use rules to stimulate an equitable division of the number of shifts and shift types.

Kennemer Gasthuis Haarlem

The planning department of the Dutch hospital Kennemer Gasthuis in Haarlem uses the optimization engine in Harmony to make non-cyclical schedules for their nursing departments. To schedule the weekend shifts they also use the Shift Sequence Planner. They use the optimization engine to assign the remaining shifts. If they do not use this workaround, the optimization engine assigns single weekend shifts and if Harmony does assign two weekend shifts in one weekend, these shifts often are of a different type (i.e. one day and one evening shift). For the Kennemer Gasthuis this workaround mostly solves their issues, but it is a labor-intensive method, which

they see as an important disadvantage.

2.3 Performance and scheduling alternatives in Harmony

Section 2.3.1 summarizes the current performance of Harmony's automatic planners. Section 2.3.2 summarizes the possibilities currently available in Harmony to schedule weekend shifts.

2.3.1 Performance

From the case study we derive four major problems with respect to scheduling weekend shifts in Harmony. The optimization engine does not always:

1. schedule all weekend shifts, whereas a feasible solution does exist for all weekend shifts.
2. divide the weekend shifts equitably over all employees: not all employees work the same number of shifts and shift types during weekends.
3. assign either zero or two shifts in a weekend.
4. assign two shifts of the same type in a weekend.

Of course, it is interesting to know why the optimization engine is not able to meet the requirements above. Especially the first problem is interesting, as a feasible solution does exist and the optimization engine does not even have to take requirements 2 to 4 into account here. The cause of this problem lies in the way the algorithm is set up. The first phase of the genetic algorithm constructs an initial population in a greedy way using the Vacant Shifts Planner. The Vacant Shifts Planner starts with assigning shifts on day 1 of the schedule, then day 2, and so on until the last day of the schedule. Of course, on the first weekend it is easy to assign all available shifts to employees, as for all employees the schedule is still empty. Nevertheless, by the end of the schedule it becomes harder to assign the available shifts to employees, as they all already have shifts that are not necessarily optimal in their schedule. The schedules resulting from this phase are thus often schedules with unassigned shifts in the last weekend(s) of the schedule. The start solution for the local search is thus of insufficient quality. The unassigned shifts cannot be assigned to an employee using the Vacant Shifts Planner (otherwise they would have been assigned in the first phase of the genetic algorithm), this means that a 1-opt change is not possible for assignment of these shifts. A 2-opt change is not helpful either, as this would mean changing an unassigned shift with an assigned shift, resulting in the same number of unassigned shifts.

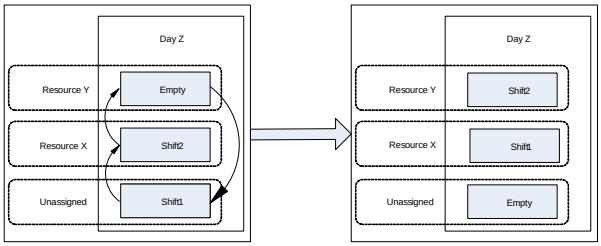

Figure 2.4: A 3-opt change in which an unassigned shift gets assigned

A 3-opt change for example, could give a better result, as Figure 2.4 shows, but unfortunately Harmony's local search does not include 3-opt. Because of the quality of the start solution for the local search, the local search has to make a too large improvement.

Problems 2, 3, and 4 are problems the optimization engine simply does not take into account if the user does not define soft constraints for these subjects. In the three cases described in Section 2.2 the users did define the corresponding constraints, but as they are soft constraints, Harmony can, and obviously does, violate them. This shows that solving problem 2, 3, and 4 by defining soft constraints is not sufficient for the users of the optimization engine.

2.3.2 Alternatives to schedule weekend shifts in Harmony

The previous sections show that the performance of the optimization engine is not sufficient for weekend shifts. Scheduling all shifts in one step is too complicated. Fortunately, the optimization engine as explained above is not the only possibility in Harmony to plan weekend shifts. This section summarizes the current possibilities in Harmony to schedule weekend shifts.

We have seen in the case study that these three customers schedule weekend shifts apart from the rest of the planning period. Weekend shifts are then scheduled as an a priori step, followed by the optimization engine for the remaining shifts. We could argue that the used sequence of scheduling decreases the flexibility for non-weekend shifts. Tests on the three cases have shown (see Table 2.1) that the reversed order (first non-weekend shifts followed by weekend shifts) on average increases the quality of the non-weekend shifts by only 6.4%. The quality is measured as the sum of all penalties on soft constraints that the users have defined. The reversed order also leaves on average 40% of the weekend shifts unassigned, where scheduling weekend shifts as a priori step leaves on average only 1.5% of all non-weekend shifts unassigned. In both sequences, the first step has no remaining shifts. Summarized, scheduling weekend shifts as a priori step only slightly decreases the flexibility of non-weekend shifts, but the total number of unassigned shifts is minimized in comparison with non-weekend shifts as a priori step.

	Calgary Health Region	Belgian Police	Kennemer Gasthuis	**Average**
Penalty increase with weekend shifts as a priori step	12%	4.6%	2.5%	**6.4%**
Remaining non-weekend shifts with weekend shifts as a priori step	1.7%	1.5%	1.3%	**1.5%**
Remaining weekend shifts with non-weekend shifts as a priori step	40%	29%	50%	**40%**

Table 2.1: Comparison of results for weekend shifts as a priori step to non-weekend shifts as a priori step

Unassigned shifts on weekend days are of more influence to a company than unassigned shifts on non-weekend days. It is hard (nobody wants to give up a free weekend) and expensive (pay irregularity allowances) to get these unassigned shifts filled. Next, the assignment of weekend shifts has the most influence on the social life of employees. To keep your employees satisfied, it is important to schedule weekend shifts as much as possible according to employee preferences.

We thus conclude that a two-step approach with scheduling weekend shifts as an a priori step is the best scheduling method. For this a priori step Harmony has several options:

- *Optimization engine for weekend shifts only*

  In this alternative the user runs the optimization engine only for weekend shifts. The advantage of this option compared to planning all shifts at the same time is that weekend shifts get preference over non-weekend shifts. This results in a better assignment of weekend shifts compared to using the optimization engine for all shifts at the same time. On the other hand, this option is not forced to schedule consecutive shifts in a weekend. This results in single weekend shifts and in consecutive shifts of different shift types. The user can define criteria to stimulate consecutive shifts of the same shift type, to prevent single weekend shifts, and to stimulate an equitable division of shifts. These criteria are soft, so the optimization engine is not forced to meet the criteria and can prefer to schedule another shift. The optimization engine for weekend shifts also faces the problem of not being able to assign all shifts to resources as described above for the optimization engine in general. Although, this option might result in fewer remaining shifts, because only weekend shifts are taken into account, but the problem can still arise.

- *Optimization engine for weekend shifts only with non-weekend rules and criteria turned off*

  The user defines rules and criteria for a schedule. Some of these rules and criteria do not concern weekend shifts. Although these rules do not directly concern weekend shifts, the planner checks on them. It might have a positive influence on the resulting schedule when these rules are turned off during the assignment of weekend shifts. Further advantages and disadvantages are equal to the option above.

- *Shift sequence planner*

  The user predefines shift sequences. For weekend shifts they start on Friday or Saturday and consist of 2 or 3 consecutive shifts. The shift sequence planner tries to assign as many shift sequences as possible to resources. The user can predefine the priority of a sequence and whether that sequence is valid for the whole department or only for selected employees. The advantage of this option is that for weekends only sequences are scheduled that the user wants, so no single weekend shifts and no consecutive shifts of different shift types (unless the user prefers so). The disadvantage is that this option does not try to divide the shifts equally over all employees. The user can define criteria to stimulate an equitable division, but again, these are soft criteria and thus not always met.

- *Plan manually*

  A last option to improve a schedule is to tweak manually. Users often (if not always) make manual adjustments after using one of the above described options to improve the resulting schedule and adjust it to their preferences.

2.4 Requirements for Weekend Planner

This section discusses which requirements the Weekend Planner has to satisfy. These requirements can be split into two parts: general requirements, resulting from within ORTEC, and requirements for the resulting schedules of the Weekend Planner, which follow from the users of the optimization engine.

The main general requirements for the Weekend Planner are:

- *Implementation*: it has to be possible to implement the solution in the current structure of Harmony.

- *Generic:* over 200 customers use Harmony and they all have to be able to work with the Weekend Planner. Aside from the three customers from the case studies, the consultants'

perception is that the problem occurs at many customers. This emphasizes the need for a generic solution.

- *No workarounds:* as described in Section 2.2 several users of the optimization engine currently use workarounds to create reasonable schedules. Workarounds are available in Harmony, but as the problem is generic, it is important to come up with a solution that makes workarounds unnecessary. The Weekend Planner finally has to become part of a one-step-scheduling process for the user.

The main requirements for the resulting schedules of the Weekend Planner are:

- All weekend shifts assigned to employees, if a feasible solution exists;
- Weekends divided equitably over all employees;
- No stand-alone weekend shifts;
- The same shift-type for all shifts in one weekend.

2.5 Related research

Burke et al. (2004) overview papers that are written on personnel scheduling problems in health care. The third section discusses both exact as well as approximation solution methods for personnel scheduling problems. Main conclusions of this article are that exact solution methods are not an option in practice, because they cannot cope with the enormous search space for real life problems, but they can form a good starting point and can aid heuristic methods. The current state of the art is represented by interactive approaches that incorporate problem specific methods and heuristics together with powerful meta heuristics, constraint based approaches, and other search methods. Very few of the presented solutions are suitable for directly solving difficult real world problems. Many of the discussed models are too simple to be directly applied to e.g. hospital wards. The solutions that are suitable for solving real world problems pose many restrictions on the problems they can solve (such as maximum instance size, existence of specific constraints, etc.).

Chapter 3

Modeling

This chapter describes the mathematical model of the problem discussed in this research. Section 3.1 gives the assumptions of the model, Section 3.2 describes the mathematical problem definition, followed by a quadratic integer programming model in Section 3.3.

3.1 Modeling assumptions

As with every translation of real life situations into a mathematical model, several assumptions have to be made. Assumptions can also be useful to decrease the problem size and to keep the problem from becoming too complicated. This section describes the assumptions we make for the problem as described in Chapter 2.

- we only consider the planning of weekend shifts in this research. This planning of weekend shifts is a pre-processing of the planning of the complete schedule;
- all employees cost the same, i.e. wages are not taken into account;
- demand for shifts is equal on Saturdays and Sundays;
- in the model we use weeks as indices for time, the week represents a complete weekend;
- if a resource works in a certain week in the model, he works both on Saturday and on Sunday and the same shift type on both days;
- a shift is a period of work, for example a morning assignment from 7 AM - 3 PM, or an afternoon assignment from 1 PM - 9 PM etc.

We realize that the above list is not complete; factors like contract hours and age related requirements are not taken into account in this stage of the research. We will discuss some important possible extensions of the model in Section 4.4.2.

3.2 Mathematical problem definition

This section starts by introducing the sets we use in the remainder of this research:

Sets

R	**resources (index** r**)**
W	**weeks (index t)**
W^+	**subset of** $W, = \{t \geq 0\}$
S	**shift types (index** s**,** z**)**

In the remainder of this research we will use *shift* as a unique combination of a week and a shift type:

Definition 3.1. Shift(s,t)
Shift(s,t) is the unique combination of shift type s in week t.

We define our Weekend Scheduling Problem (WSP) in Definition 3.2.

Definition 3.2. (Weekend Scheduling Problem)
GIVEN: A finite set of time periods W, a set of resources R, a set of shift types S, the following parameters:

d_{ts} demand: number of resources needed for *shift(s,t)*

a_{rts} $\begin{cases} 1 & \text{if resource } r \text{ is available for } \textit{shift(s,t)} \text{ (vacation, skills, etc.)} \\ 0 & \text{otherwise} \end{cases}$

we_r maximum number of weekends resource r can work in w_r weeks

w_r number of weeks in which resource r can work we_r weekends,

q_{rt} $\begin{cases} 1 & \text{if resource } r \text{ works in week } t \ (t<0) \\ 0 & \text{otherwise} \end{cases}$

and the following decision variables

X_{rts} $\begin{cases} 1 & \text{if resource } r \text{ works shift type } s \text{ in week } t \\ 0 & \text{otherwise} \end{cases}$ $t = 0, \ldots, W$

C_{rts} $\begin{cases} 1 & \text{if resource } r \text{ is allowed to work shift type } s \text{ in week } t \\ 0 & \text{otherwise} \end{cases}$

GOAL: Does there exist a binary vector $x = (X_{rts})$, such that the deviation between the number of weekends a resource works and the average workload per resource, and the deviation between the number of shifts a resource works of each allowed shift type is minimized (an equitable division of weekend assignments over the resources) for cyclical schedules (variant a) and non-cyclical schedules (variant b) and such that the following constraints, of which the mathematical formulation is given in Section 3.3, hold:

- demand d_{ts} is satisfied in every week t for every shift type s (constraint 3.4);
- every resource works at most one shift type per week (constraint 3.5);
- every resource only works shifts that it is allowed to work (constraint 3.6);
- every resource works maximum we_r weekends in w_r weeks (constraint 3.8a for cyclical schedules and constraint 3.8b for non-cyclical schedules).

The next section presents a formulation to solve this Weekend Scheduling Problem.

3.3 Quadratic Integer Programming model

This section describes a quadratic integer programming formulation (QIP) of WSP (Definition 3.2). First, we discuss the formulation of the objective function, followed by the formulation of the complete model and the explanation of the constraints.

Objective function

The objective function consists of two parts. The first part establishes a minimization of the difference between the number of weekends a resource works and the average workload per resource. This difference is taken quadratically because a deviation of 4 for one resource is worse than a deviation of 1 for four resources for example. This first part of the objective is as follows:

$$\min \sum_{r} \left(\sum_{t,s} X_{rts} - \frac{\sum_{t,s} d_{ts}}{|R|} \right)^2 \tag{3.1}$$

The second part of the objective function consists of the minimization of the deviation from an equitable division of shift types for each resource. The goal of this objective is to make sure that each resource works approximately the same number of weekends of each shift type that that resource is allowed to work. For example, two resources working both one weekend with day shifts and one weekend with night shifts is preferred over one resource with two weekends of night shifts and the other resource working two weekends of day shifts.

This deviation is formulated as the difference in the number of weekends worked between all possible combinations of shift types s and z for each resource. We only take the value of a worked weekend with shift type s into account if the resource would have been allowed to work shift type z as well. To accomplish this, we multiply variable X_{rts} with the parameter for availability for shift type z: a_{rtz} and vice versa. To make sure we take all possible combinations of shift types into account, we compare for each resource the number of weekends worked with shift type s to the number of weekends worked with shift type z, if the ordinality of z is larger than the ordinality of s. As the deviation consists of positive and negative deviation, we need to take the absolute value:

$$\min \sum_{r} \sum_{s,z|ord(s)<ord(z)} \left| \sum_{t} (a_{rtz} \cdot X_{rts} - a_{rts} \cdot X_{rtz}) \right| \tag{3.2}$$

To convert the above objective into a linear formulation, we first introduce two assisting variables:

z^+_{rsz} Assisting variable for the positive deviation from an equitable division of shift types s and z for resource r

z^-_{rsz} Assisting variable for the negative deviation from an equitable division of shift types s and z for resource r

Next, we introduce two extra constraints to make sure that z^+_{rsz} and z^-_{rsz} represent the positive respectively negative deviation:

$$z^+_{rsz} \geq \sum_t (a_{rtz} \cdot X_{rts} - a_{rts} \cdot X_{rtz}) \qquad \forall r,\ s,\ z$$

$$z^-_{rsz} \geq \sum_t (a_{rts} \cdot X_{rtz} - a_{rtz} \cdot X_{rts}) \qquad \forall r,\ s,\ z$$

and to make sure that for each resource only one of the two assisting variables gets a nonzero value:

$$z^+_{rsz} \geq 0 \qquad \forall r,\ s,\ z$$

$$z^-_{rsz} \geq 0 \qquad \forall r,\ s,\ z$$

Now we can replace formula (3.2) by the following linear minimization:

$$\min \sum_r \sum_{s,z|ord(s)<ord(z)} \left(z^+_{rsz} + z^-_{rsz}\right)$$

The two parts of the objective function are not equally important. The equitable division of the number of shifts is most important. An increase in penalty for the division of the number of shifts over an equal decrease in penalty for the division of shift types should be prevented. That is why we give both parts of the objective function a weight, α (for the part on the division of the number of shifts) and β (for the part on the division of shift types), where α is larger than β.

QIP formulation

The preceding sections lead to the following QIP formulation for the Weekend Scheduling Problem:

$$\min \sum_r \left(\alpha \cdot \left(\sum_{t,s} X_{rts} - \frac{\sum_{t,s} d_{ts}}{|R|} \right)^2 + \beta \cdot \sum_{s,z|ord(s)<ord(z)} \left(z^+_{rsz} + z^-_{rsz}\right) \right) \tag{3.3}$$

$$\text{s.t.} \quad \sum_r X_{rts} \geq d_{ts} \qquad \forall t,\, s \tag{3.4}$$

$$\sum_s X_{rts} \leq 1 \qquad \forall r,\, t \tag{3.5}$$

$$X_{rts} \leq Y_{rts} \qquad \forall r,\, t,\, s \tag{3.6}$$

$$C_{rts} \leq a_{rts} \qquad \forall r,\, t,\, s \tag{3.7}$$

$$\sum_{\tau=t}^{(t+w_r-1)\bmod|W|} \sum_s X_{r\tau s} \leq we_r \qquad \forall r,\, t \tag{3.8a}$$

$$\left(\sum_{\tau=t}^{(t+w_r-1)} (\sum_s X_{r\tau s}) + q_{r\tau} \right) \leq we_r \qquad \forall r,\, t = -w_r+1, \ldots, \left|W^+\right| - w_r + 1 \tag{3.8b}$$

$$z^+_{rsz} \geq \sum_t (a_{rtz} \cdot X_{rts} - a_{rts} \cdot X_{rtz}) \qquad \forall r,\, s,\, z \tag{3.9}$$

$$z^-_{rsz} \geq \sum_t (a_{rts} \cdot X_{rtz} - a_{rtz} \cdot X_{rts}) \qquad \forall r,\, s,\, z \tag{3.10}$$

$$z^+_{rsz} \geq 0 \qquad \forall r,\, s,\, z \tag{3.11}$$

$$z^-_{rsz} \geq 0 \qquad \forall r,\, s,\, z \tag{3.12}$$

$$X_{rts} \in \{0,1\} \qquad \forall r,\, t,\, s \tag{3.13}$$

Constraints (3.4)-(3.8b) describe the constraints as given in Definition 3.2. Constraint (3.4) enforces that the demand is met. Constraint (3.5) forces a resource to work maximal one shift per day. Constraint (3.6) assures that a resource only works the types of shifts it is allowed to work. Constraint (3.7) makes sure that resources are only allowed to work a shift if they were available for that shift according to parameter a_{rts}. Constraints (3.8a)/(3.8b) make sure that a resource does not work more weekends than the maximum number it is allowed to work by contract. Constraints (3.9) and (3.10) are the extra constraints to eliminate the absolute value from the objective function.

Chapter 4

Weekend Planner design and description

This chapter proposes a solution to solve the WSP: the Weekend Planner. Section 4.1 explains some of the terminology used. Section 4.2 discusses the main considerations in designing the Weekend Planner. Finally, Section 4.3 describes the proposed heuristic.

4.1 Terminology

In this chapter we use the definitions of *weeks* and *shift types* as used earlier in this research. So by a *week* we mean a complete weekend. By a *shift(s,t)* we mean the unique combination of a week and a shift type. A *resource* is an employee who can work shifts. The goal is to assign shifts to resources according to the WSP with its constraints as described in Chapter 3.

4.2 Weekend Planner design

This section summarizes the main insights for solution design from Chapter 2. Creating a schedule in one integral step is too complicated (as results in Section 5.6.2 also show). Users of Harmony's optimization engine currently schedule in two sequential steps. The best scheduling sequence has shown to be scheduling weekend shifts as a priori step (see Section (2.3.2)), followed by the remaining, non-weekend shifts. For the a priori step various methods are used, ranging from manual scheduling to using one of Harmony's automatic planners or a combination of methods. They schedule the remaining shifts with Harmony's optimization engine. This two-step approach

results in a satisfying schedule according to users, but the a priori step is labor-intensive. As Harmony's optimization engine is suitable for the scheduling of the non-weekend shifts, we focus on a new method for the a priori step: the Weekend Planner.

The Weekend Planner has to take the following objectives into account in lexicographic order of importance:

1. create an assignment of weekend shifts with zero remaining shifts (if possible)
2. create an assignment of weekend shifts with an equitable division of the number of weekends worked
3. create an assignment of weekend shifts with an equitable division of shift types

Because of the way we have modeled the WSP, the Weekend Planner automatically takes the following two requirements, which also follow from Chapter 2, into account:

- create an assignment of weekend shifts without stand-alone weekend shifts
- create an assignment of weekend shifts with consecutive shifts of the same shift type

As the current possibilities in Harmony do not focus on the above objectives, this research proposes a solution that is not based on the current structure of the available algorithms in Harmony. We propose a heuristic approach, the Weekend Planner, which uses an intelligent form of list scheduling to select shifts and to assign them to resources. The Weekend Planner assigns *shift(s,t)* to an employee, which means that if an employee works *shift(s,t)*, the employee works shift type s on Saturday and Sunday of the corresponding week t. List scheduling takes objective 2 into account if the selection rules are set up accordingly. To meet objective 1, the Weekend Planner will schedule the least flexible shifts and the least flexible resources first. List scheduling does not take objective 3 into account. Therefore the last step of the Weekend Planner is a local search with the objective to improve the equality of the division of shift types.

4.3 Weekend Planner description

This section describes the following steps of the Weekend Planner (Algorithm 4.1) for one solution in more detail:

- Select shift (Section 4.3.1)

Algorithm 4.1 Weekend Planner

```
for Number of Samples
    while (remainingShifts > 0) and (availableResources > 0)
        Select shift
        Select resource
        Assign shift to resource
        Update availability
        Check weekend constraint
    end
    Local search
    if Solution is better than BestSolutionSoFar
        BestSolutionSoFar := Solution
    end
end
```

- Select resource (Section 4.3.2)
- Assign shift to resource (Section 4.3.3)
- Local search (Section 4.3.4)

Because of randomness in the Weekend Planner, we apply a sampling technique, which we explain in Section 4.3.5.

The Weekend Planner uses selection rules for the construction of a solution. To make sure that the largest number of shifts is scheduled, it is important to schedule the least flexible shifts and the least flexible resources first. The last shifts to schedule are the shifts with the most flexibility. Flexibility depends on the total number of comparable shifts that still have to be scheduled, and on the number of resources that are still available for that shift. The least flexible shift is assigned to the least flexible resource. Flexibility of resources depends on the number of shifts already assigned to a resource in the planning period and on the number of shifts a resource is available for. We only take feasible shift-resource combinations into consideration.

The Weekend Planner uses four rules to select a *shift(s,t)* and four rules to select a resource. If the first rule results in a tie, the Weekend Planner uses the second rule, etc. However, some selection rules calculate a ratio for each *shift(s,t)* or for each resource. A ratio will hardly ever result in a tie, which means that the Weekend Planner probably does not frequently use the succeeding selection rules. That is why we use a parameter k for precision. We test various values for k in Chapter 5 to determine the best scenario. In both the selection of a *shift(s,t)* and a resource, the last selection rule is based on randomness. To test the influence of randomness, Chapter 5 also analyzes how often the Weekend Planner uses each selection rule.

4.3.1 Select shift

The Weekend Planner uses the following rules in decreasing order of importance to select *shift(s,t)*:

1. *Shift(s,t) with the lowest flexibility ratio: "unfulfilled demand for shift(s,t)/ remaining available resources"*

$$\textit{shift(s,t) for which } \frac{\left(\sum_r X_{rts} - d_{ts}\right)}{\sum_r Y_{rts}} \textit{ is minimal.}$$

   The shift that this rule selects is the least flexible shift, because for this shift less resources are available than for all other shifts. So this shift is the hardest shift to plan and for this reason has to be assigned to a resource first.

   This rule results in a tie when two ratios are equal. Note: both are rounded on k decimals; k is an experimental factor.

2. *Shift(s,t) with the smallest number of shifts still to be scheduled:*

$$\textit{shift(s,t) for which } \left(d_{ts} - \sum_r X_{rts}\right) \textit{ is minimal.}$$

   If the first rule results in multiple possible shifts, the *shift(s,t)* with the smallest absolute number of shifts to be scheduled is selected. For example: take *shift(A,1)*(shift type A in week 1) and *shift(B,1)*(shift type B in week 1). For *shift(A,1)* still 2 shifts have to be assigned and there are 4 available resources. For *shift(B,1)* still 20 shifts have to be assigned and there are 40 available resources. Both shifts have a flexibility ratio of -0.5, but assigning *shift(A,1)* is less flexible than assigning *shift(B,1)*, so *shift(A,1)* is selected.

3. *Select first shift in time line*

   From the remaining shifts, this rule selects the shift that is closest to the start of the planning period. This is important because resources have a history, so the weeks closest to the previous planning period are more influenced by this history than weeks more to the end of the planning period. Shifts close to the start of the planning period are less flexible than shifts later in the planning period, due to more constraints on the number of weekends that still can be worked.

4. *Select random shift*

   If the first three rules result in multiple possible shifts, a random shift is taken.

4.3.2 Select resource

The Weekend Planner uses the following four rules in decreasing order of importance to select a resource for the selected *shift(s,t)* :

1. *"Least busy resource": resource with lowest number of weekends scheduled in scheduling period:*

$$\textit{resource for which } \sum_{t,s} X_{rts} \textit{ is minimal.}$$

   This rule makes sure that each resource is assigned to approximately the same number of weekends, as is stated in the objective function of the WSP. The resource with the lowest number of assigned shifts is chosen.

2. *"Least flexible resource": resource with lowest number of available shifts that the resource is allowed to work :*

$$\textit{resource for which } \sum_{t,s} Y_{rts} \cdot \left(d_{ts} - \sum_{r} X_{rts} \right) \textit{ is minimal.}$$

   A resource that is allowed to work ten of the available shifts (shifts still to be scheduled) is more flexible than a resource that is allowed to work only four available shifts. The number of shift types a resource is allowed to work influences this rule, but also the number of weeks a resource has requested vacation for. This rule selects the least flexible resource.

3. *Resource for which the flexibility ratio of the least flexible other* shift(s,t) *that that resource is allowed to work in the same week* t *as the selected* shift(s,t)*, is maximal.* This rule finds for each resource the least flexible shift that that resource is allowed to work in week *t*, except for the selected shift. This rule then selects the resource(s) for which the least flexible shift is most flexible (of all found least flexible shifts). This is thus the resource for which the other shifts that that resource can work, have the most flexibility, so the Weekend Planner does not lose much flexibility with assigning the selected shift to this resource. The other shifts this resource could work in the same weekend are the 'easiest' to plan, as their ratio is maximal. This rule selects:

$$\textit{resource for which } \min_{s|Y_{rts}=1} \frac{\left(\sum_{r} X_{rts} - d_{ts} \right)}{\sum_{r} Y_{rts}} \textit{ is maximal for given } \text{t} \textit{ of selected } \text{shift(s,t)}.$$

   For example, consider selected *shift(A,1)*, for which two resources 'compete'. Resource 1 is allowed to work type *shift(A,1)* and *shift(B,1)*. Resource 2 is allowed to work type *shift(A,1)*

and *shift(C,1)*. The flexibility ratio for *shift(B,1)* is -0.3 and for *shift(C,1)* -0.2. *Shift(B,1)* is thus harder to assign than *shift(C,1)*. So resource 1 is more 'needed' for *shift(B,1)* than resource 2 is 'needed' for *shift(C,1)*. It is more flexible to assign *shift(A,1)* to resource 2 in this step.

4. *Select random resource*

   If the above rules result in multiple resources, select a random resource from the remaining resources.

4.3.3 Assign shift to resource

Once a resource has been selected, the selected shift is assigned to the selected resource. The availability variable C_{rts} is updated for resource r after every assignment of *shift(s,t)* to that resource. The availability for the selected resource is set to 0 for all shift types in the week of the selected shift. Next, the algorithm performs a check on the number of weekends a resource works to check on the weekend constraint (constraints 3.8a/3.8b in Section 3.3) and adjusts the availability parameter if necessary.

The process of selecting shifts and resources continues until all shifts have been assigned to resources, or until no more shifts can be assigned to resources.

4.3.4 Local search

The Weekend Planner stimulates an equitable division of the number of shifts. Thus far, it does not focus on an equitable division of shift types. Therefore, we perform a 2-opt local search to improve the division of shift types. The 2-opt local search included in the Weekend Planner evaluates shift swaps for each combination of two assigned shifts in the same week. The swap is made if it results in a lower value of the objective function and thus in a more equitable division of shift types (Algorithm 4.2).

The local search iterates a given number of times over all employees to find improvements. The number of iterations is bounded by a maximum, but stops earlier when no improvements are found in the last iteration. Chapter 5 analyzes various values for the maximum number of iterations to determine the best scenario.

Algorithm 4.2 2-opt Local Search included in the Weekend Planner

```
While (improvement found in last iteration) and (NoIterations<MaxIterations)
    for all Weeks
        for every 2 resources that work in the same week, but different
        shift types
            if resources are allowed to work each others shift type
                Exchange shifts and calculate new score
                    if new score < current score then keep exchange
                    else change shifts back
                end
            end
        end
    end
end
```

4.3.5 Sampling

As there is a random component in the construction algorithm, not every solution for an instance has to be the same. One solution might be better than another, generated with other random values. The Weekend Planner creates a solution a number of times (as requested by the user) and selects the best solution to give back as output. This process is called *random sampling* (Kolisch and Drexl, 1996). As randomness only affects the last selection rule of both shift and resource, the influence of randomness and thus of sampling might be small. Chapter 5 addresses the influence of randomness and analyzes various number of samples to determine the best scenario.

4.4 Alternative algorithm designs and model extensions

This section discusses some alternative designs of the Weekend Planner (Section 4.4.1). Next, the Weekend Planner is a generalized form of the cases that occur in practice. Section 4.4.2 summarizes a few extensions that frequently occur and the way they can be included in the Weekend Planner.

4.4.1 Alternative algorithm designs

Take random shift in time

An option for alternative design is to exclude the selection rule *select first shift in time line*. An argument to exclude this rule is that this rule always favors the start of the planning period.

This could result in problems towards the end of the planning period. Chapter 5 addresses the consequences of excluding this rule.

Regret-based random sampling

Regret-based random sampling (RBRS) is a randomized construction heuristic often used in scheduling, proposed by Kolisch and Drexl (1996). This method uses priority rules to value solution building blocks. In the Weekend Scheduling Problem the shifts and resources are the solution building blocks, and for example the flexibility of a shift determines the priority of that shift. Instead of simply selecting the building block with the highest priority, this method calculates a probability for each building block based on its priority. The higher the priority, the higher the probability of a building block is. RBRS then selects a building block from all candidates, using their probabilities. This method also uses sampling (see Section 4.3.5).

This method calculates priority v_j for building block j using a priority rule. The user defines the priority rule. Given a priority v_j, we calculate the regret factor r_j, which is the non-negative difference of the priority v_j and the worst of all building block priorities:

$$r_j = v_j - \min_i \{v_i\}, \text{ if a high } v \text{ implies a high priority}$$

$$r_j = \max_i \{v_i\} - v_j, \text{ if a high } v \text{ implies a low priority}$$

The higher the regret factor r_j is, the more regret you have of not selecting the building block. Based on the regret factor we calculate the probability P_j of a solution building block being drawn:

$$P_j = \frac{(r_j + 1)^\gamma}{\sum_i (r_i + 1)^\gamma} \tag{4.1}$$

The parameter γ is the so-called bias-factor. the larger γ is chosen, the larger the effect of the regret factor is.

As an alternative design, we apply regret-based random sampling to the Weekend Scheduling Problem, based on the selection rules of the Weekend Planner. We adjust the described general form of RBRS to be suitable for the WSP. We calculate a separate probability for the selection of a shift and for the selection of a resource. We here describe the method to select a shift, the same holds for resource selection. The probability in the standard RBRS is based on only one priority rule. The Weekend Planner uses various priority rules to select a shift. We calculate the regret and probability per shift per priority rule as in the general RBRS. This results in three probabilities P_i for each shift. We define a weight w_i for each priority rule i, such that the weights

of shift selection sum up to 1. We then multiply each weight with the corresponding probability and sum these values for all three priority rules:

$$P_s = \sum_i w_i \cdot P_i$$

Each shift then has one probability P_s, based on which a shift is selected. The priorities of all shifts obviously sum up to 1.

The standard formula to calculate a probability (formula 4.1) increases the regret with 1, such that each building block has a nonzero probability of being drawn. The Weekend Planner contains selection rules in which priority and regret are based on the flexibility ratio of shifts. The regret is thus always < 1. Increasing the regret with 1 would destroy the proportion between the probabilities for different shifts, as this increase is larger than the regret itself. For priority rules based on ratios (the first shift selection rule and the third resource selection rule) we increase the regret with 0.01 instead of 1 preventing a destruction of proportions, but maintaining a nonzero probability for each building block:

$$P_j = \frac{(r_j + 0.01)^\gamma}{\sum_i (r_i + 0.01)^\gamma}$$

The selection rules that the Weekend Planner uses are defined as consecutive rules. RBRS uses priority rules as independent rules. Consecutive rules are not necessarily suitable for independent use. The second shift selection rule selects the shift(s,t) with the smallest number of shifts still to be scheduled, *if* the first rule (select the least flexible shift(s,t)) results in a tie. Independently, the second rule does not make sense: the shift(s,t) with the smallest number of shifts still to be scheduled should not get a high priority. We thus set the weight of this priority rule for use in RBRS to 0.

4.4.2 Model extensions

FTE (Full Time Equivalent) The Weekend Planner stimulates an equitable division of the absolute number of shifts. However, when not all resources work the same FTE, users might prefer to divide the weekends in proportion to the FTE. To accomplish this, the first rule to select a resource has to be changed to : *resource for which* $\frac{\sum_{t,s} X_{rts}}{FTE}$ *is minimal.*

Night shifts In some industries, it is common that employees have to work no or a limited number of night shifts after having reached a certain age. To include not working night shifts in

the Weekend Planner, the availability a_{rts} has to be set to 0 for resource r for all t and for all shift types s that are night shifts. To include a maximum number of night shifts, an extra constraint should be added that checks the number of night shifts for an employee after each assignment of a shift (comparable with the check on the maximum number of weekends constraint).

Chapter 5

Computational results

This chapter describes the results for the Weekend Planner. Section 5.1 gives the experiment approach and a further outline of this chapter.

5.1 Experiment approach

The experiment consists of the following steps:

1. Determine the best scenario of the Weekend Planner:
   (a) Test Weekend Planner on a diverse set of random generated instances (Section 5.2 and 5.3)
   (b) Analysis of the sensitivity of the Weekend Planner for various parameters (Section 5.4):
      i. Algorithm settings
      ii. Instance parameters
   (c) Test alternative designs (Section 5.5)
2. Test the best scenario of the Weekend Planner on three real-life cases from practice and compare these results with Harmony's results (Section 5.6)

5.2 Generation of random instances

For the tests we use randomly generated test instances. To generate these instances we first randomly assign shifts to employees. Second, we choose the instance parameters such that the assignment of shifts to employees is optimal. Third, we calculate the objective value for this instance.

1. *Randomly assign shifts to employees*

   We create a large number of employees and randomly assign shifts to them. In this way we have created many different employees/schedules of which we can choose some that form a feasible and optimal schedule in the next step.

2. *Choose instance parameters such that schedule is optimal*

   (a) We randomly choose instance parameters. See Table 5.1 for these parameters and the possible values. These bounds are based on values that occur often in practice. The scope is defined as the maximum number of weekends an employee is allowed to work in a period (thus defined as x out of y weeks). The tightness is defined as the ratio of the number of weekends an employee has to work over the maximum number of weekends an employee is allowed to work. From the tightness we can thus calculate the number of weekends that each employee should work if each employee works the same number of shifts.

Parameter	**Values**
Number of employees	[10..70]
Number of weeks	[4,6,8,10,13]
Scope	[1/2 , 2/4 , 3/4 , 2/5]
Tightness	[1/max ..1]
Cyclical	[true, false]
Number of shift types	[1..6]

Table 5.1: Parameter settings for testing random instances

   (b) We select the chosen number of employees randomly from the large number of employees created in step 1, such that the created schedule is feasible and optimal. Feasibility is assured by selecting employees that have a schedule for the right number of weeks, the right number of weekends worked, with the right number of shift types, and for which is checked that the schedule is feasible for the chosen scope in combination with being cyclical or not. Optimality is assured because of equality in the division of the

number of shifts and equality in the division of shift types. Equality in the division of the number of shifts is assured because every employee works the same number of weekends: the chosen value in step (a). Equality in the division of shift types is assured because only employees are selected that work the same number of shifts of every shift type (or one more of some shift types when the number of shift types is not a multiple of the number of weekends every employee works).

(c) Based on the assigned shifts, we calculate the corresponding demand. To make the instance more realistic, not all employees are available for all shifts (i.e. in case of vacation). For every shift(s,t) that an employee does not work in the created schedule, the availability of the employee for that shift is randomly set to 0 (not available) or 1 (available). The randomness of the availability is influenced by an instance parameter, a randomly chosen value between 0 and 1 that gives the probability of employees in that instance being available for a shift.

3. *Calculate objective value for instance*

   Now that all parameters are chosen such that the schedule is optimal, we can calculate the objective value. As we know that the schedule is optimal, we thus know the optimal value for this instance. The next section compares the results of the Weekend Planner with the optimal value.

5.3 Test results on random instances

This section evaluates the results of the Weekend Planner on random instances on the following performance indicators:

- *Deviation from optimal solution*

  For each instance the absolute as well as the relative deviation from the optimal solution is measured. The results also show the average over a large number of instances of the absolute and relative deviation from optimal. The deviation is the difference between the test result and the optimal solution according to formula 3.3.

- *Remaining shifts*

  For each instance the number of shifts that the Weekend Planner has not been able to assign to employees.

Setting	Value
Precision	1
Number of samples	20
Local search iterations	5
α (weight on first part objective function)	2
β (weight on second part objective function)	1

Table 5.2: Algorithm settings for tests on the Weekend Planner

We use 400 random instances, for which we know the optimal solution. The number of 400 is selected based on preliminary tests that show the average deviation from the optimal solution becoming a flat and stable line with 400 instances.

Table 5.2 illustrates the settings used to test the Weekend Planner. The values for α and β are found by tests that are not illustrated here. Section 5.4.1 analyzes the sensitivity of the Weekend Planner for the other settings.

Deviation from optimal solution

Figure 5.1 shows the absolute and relative deviation from the optimal solution for each of the 400 instances. In 67.5% of these instances the Weekend Planner has found the optimal solution.

Figure 5.2 shows the average absolute and relative deviation from the optimal solution. The average absolute deviation is 13.9; the average relative deviation is 3.8% from optimal. The figures show that for some cases the deviation from optimal is very high. This deviation is mainly caused by a non-equitable division of shift types. Figure B.1 in appendix C shows the total penalty, the penalty for an equitable division of the number of shifts, and the penalty for an equitable division of shift types. This figure shows that the graph for the total penalty is equal or quite close to the penalty for an equitable division of shift types for all instances. In the optimal solution every employee works exactly the same number of shifts, which means that the penalty for an equitable division of the number of shifts is 0. When the total penalty in the Weekend Planner is equal to the penalty for the equitable division of shift types, we know that the division of the number of shifts is optimal.

As the deviation from optimal is mainly caused by a non-equitable division of shift types, this means that all employees work the optimal number of shifts, just not the optimal shift types. We expect that the division of shift types improves by interchanging shifts between employees. The local search in the Weekend Planner already does this, but it is a simple and basic local search. A more extensive local search tries more and different shift swaps. We thus expect that a more

extensive local search, as is already available in Harmony, decreases the penalty for an equitable division of shift types and so decreases the deviation from optimal.

Remaining shifts

Figure 5.3 shows the number of remaining shifts for each instance. For 89% of the instances the Weekend Planner has found a solution with 0 remaining shifts. The average number of remaining shifts is 0.85 shift per instance (measured over all 400 instances). For some instances the number of remaining shifts is quite high. The sensitivity analysis in Section 5.4.2 analyzes the influence of the different instance parameters on the results of the Weekend Planner and the possible causes for high remaining shifts.

5.4 Sensitivity analysis

This section analyzes the sensitivity of the Weekend Planner for variation in the algorithm settings (Section 5.4.1) and for the different values of instance parameters (Section 5.4.2).

5.4.1 Algorithm settings

This section analyzes the sensitivity of the Weekend Planner for the following algorithm settings:

- Precision parameter k (Section 5.4.1.1)
- Number of samples (Section 5.4.1.2)
- Number of iterations in the local search (Section 5.4.1.3)

For each of these settings, we analyze the influence of different values for a setting on the number of instances with remaining shifts, the average number of remaining shifts per instance, the number of instances for which the Weekend Planner finds the optimal solution, and the average deviation from the optimal solution.

5.4.1.1 Precision

Figures 5.4 and 5.5 show the influence of the precision setting on the deviation from optimal and on the remaining shifts. Both figures show that the best results, although with a small

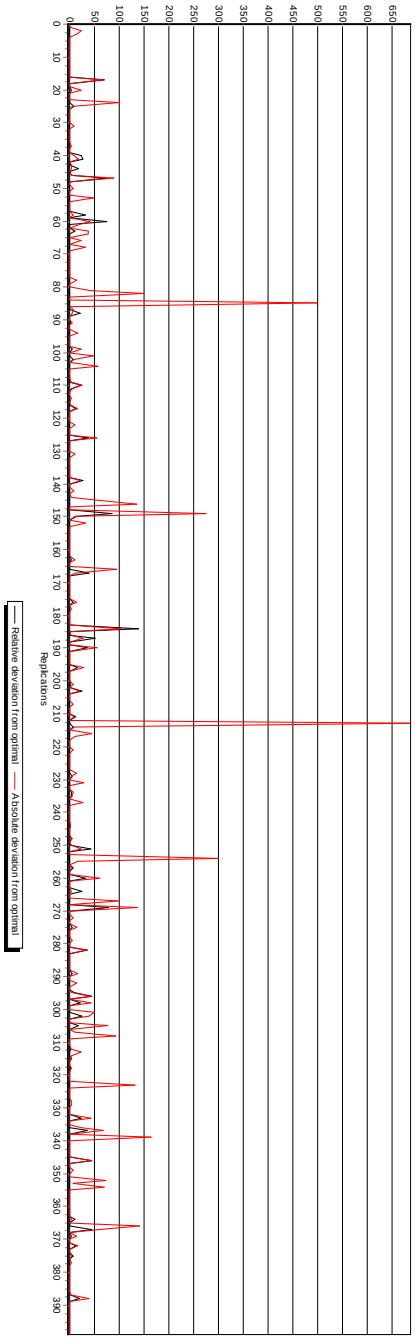

Figure 5.1: Absolute and relative deviation from the optimal solution for the Weekend Planner

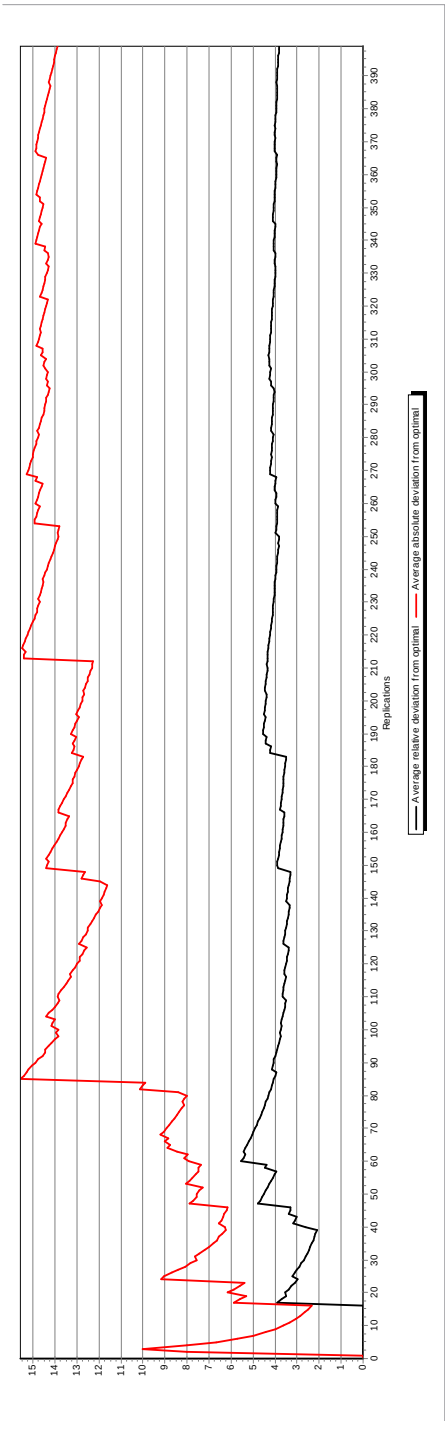

Figure 5.2: Average absolute and relative deviation from the optimal solution for the Weekend Planner

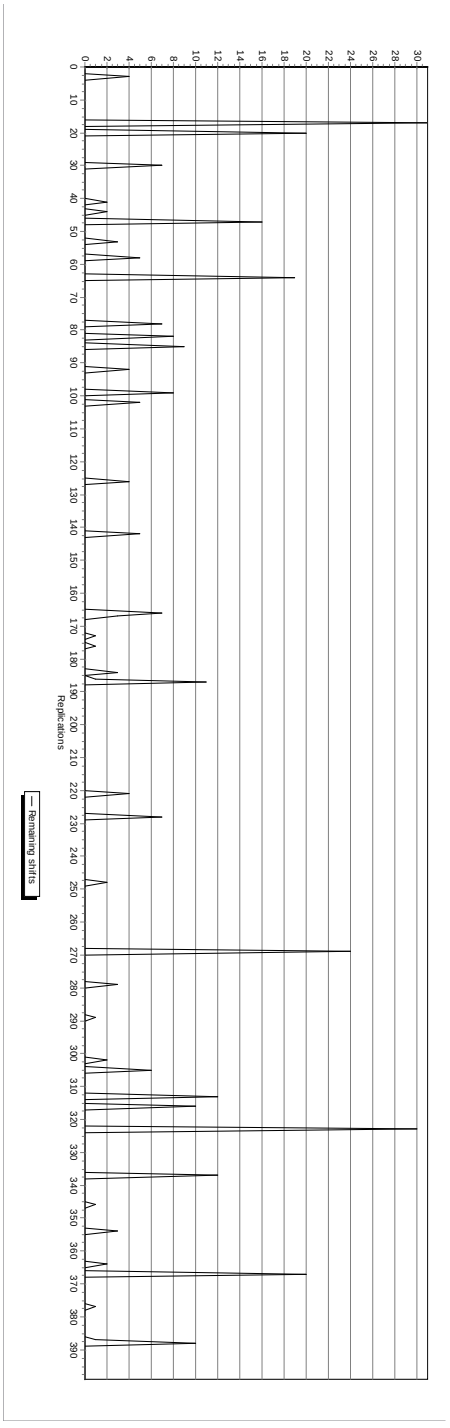

Figure 5.3: Remaining shifts

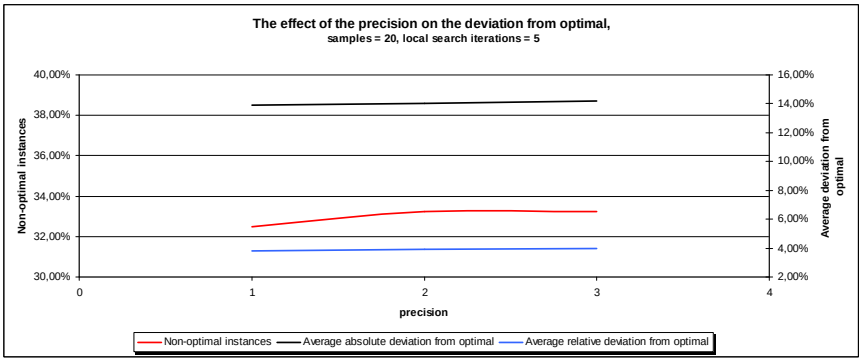

Figure 5.4: The effect of precision on the deviation from optimal

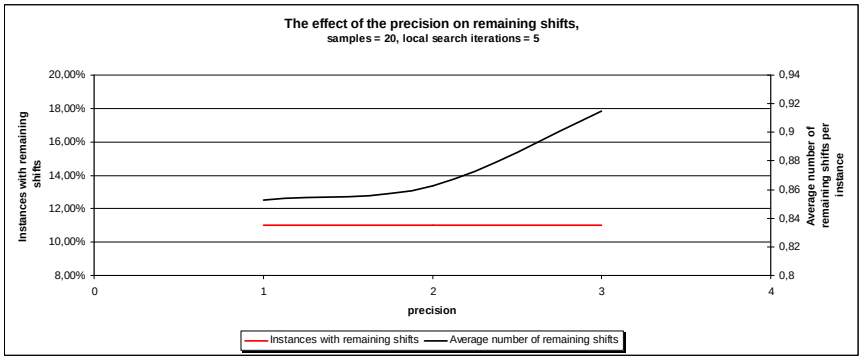

Figure 5.5: The effect of precision on the remaining shifts

difference, are achieved with a precision of 1. A precision of 1 means that the selection rules compare ratios rounded on 1 decimal. The small positive difference is explained by the influence of randomness. Randomness has more influence with a lower precision, because then a rule based on ratios more often results in ties and the random selection rules are more often called. The average usage of rules confirms this, although this difference is small too. The use of random selection rules increases from 11% (for a precision of 3) to 16% (for a precision of 1) for shift selection and from 39% to 42% for resource selection. For shift selection, the first rule is based on comparing ratios. A precision of 1 instead of a precision of 3 increases the usage of the second shift selection rule from 45% to 72%. More selection factors are thus of influence with a lower precision, which gives a slightly better result.

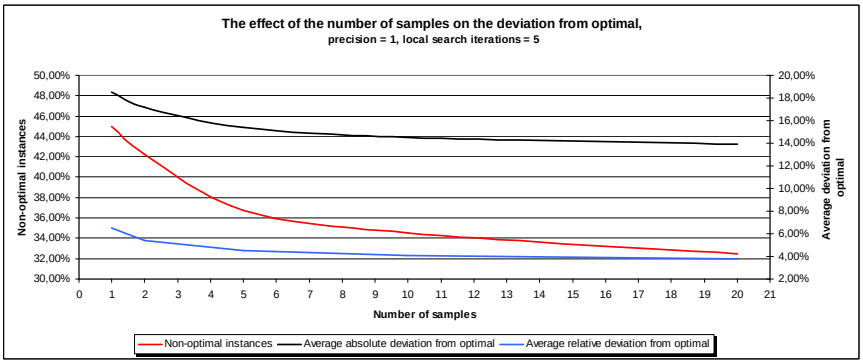

Figure 5.6: The effect of the number of samples on the deviation from optimal

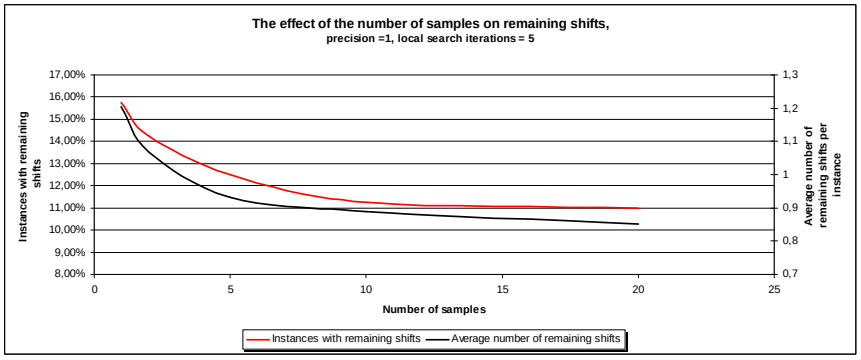

Figure 5.7: The effect of the number of samples on the remaining shifts

5.4.1.2 Number of samples

Figures 5.6 and 5.7 show the influence of the number of samples on the deviation from optimal and on the remaining shifts. The figures show that the higher the number of samples, the better the results are. We have set 20 samples as the maximum test value, because of running time. The figures also show that the largest improvement in results is achieved in the lower number of samples. An increase from 1 to 5 samples achieves more improvement than an increase from 15 to 20 samples for example. The best results are obviously achieved with the largest number of samples, in this case 20. If running time becomes an issue, then 10 or even only 5 samples give good results too.

5.4.1.3 Number of local search iterations

Figures 5.8 and 5.9 show the influence of the number of local search iterations on the deviation from optimal and on the remaining shifts. As the local search does not look at the remaining

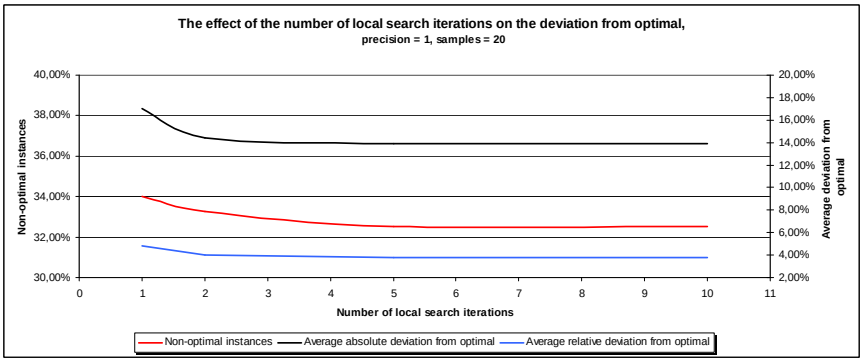

Figure 5.8: The effect of the number of local search iterations on the deviation from optimal

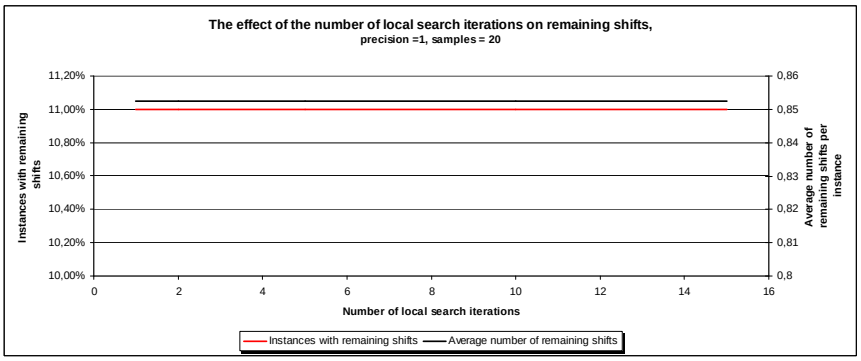

Figure 5.9: The effect of the number of local search iterations on the remaining shifts

shifts, Figure 5.9 shows a flat line. Figure 5.8 shows that the deviation from optimal improves with a higher number of local search iterations. But the figure also shows that 5 iterations is the best value, as more local search iterations do not find any more improvements.

5.4.2 Instance parameters

This section gives the conclusions on the sensitivity of the Weekend Planner for various instance parameters, which are analyzed in Appendix C in detail:

- *Instance size*: number of employees and number of weeks (Appendix C.1)
- *Availability*: probability that employees are available for shifts (Appendix C.2)
- *Scope*: the number of weekends an employee is allowed to work in a certain period (i.e. x out of y weekends) (Appendix C.3)

- *Tightness of schedule*: the average number of weekends per employee divided by the maximum number of weekends an employee is allowed to work (Appendix C.4)
- *Cyclical*: whether it concerns a cyclical or a non-cyclical schedule (Appendix C.5)

For each of these parameters, we analyze the influence on the deviation from the optimal value (relative and absolute) and on the number of remaining shifts. Section 5.4.2.1 concludes on the sensitivity of single parameters. Section 5.4.2.2 analyzes the influence of interesting combinations of the above listed parameters that follow from the analysis per parameter.

5.4.2.1 Sensitivity of single parameters

This section shortly concludes on the results on the Weekend Planner's sensitivity for the various parameters. The first paragraph gives a conclusion on the deviation from optimal, followed by a conclusion on the remaining shifts.

The deviation from optimal is mainly influenced by the length of the schedule, the availability, and the tightness of the schedule: all factors that clearly increase the complexity of the instance. As said before, the deviation mainly consists of a penalty for a non-equitable division of shift types. Therefore we expect that the deviation decreases when the Weekend Planner uses a more extensive local search.

Parameters that influence the number of remaining shifts are the scope for the maximum number of allowed weekends to work, the tightness of the schedule, and whether a schedule is cyclical or not. The number of weeks has an influence as well, although this influence seems to have an interaction with other parameters, as there is no clear increase or decrease in remaining shifts visible.

The Weekend Planner is mainly sensitive cyclical schedules and schedules with a tightness of 1. Next to that, we expect that the Weekend Planner is highly sensitive for certain combinations of schedule lengths and scopes when regarding the number of remaining shifts. We will investigate this in Section 5.4.2.2.

5.4.2.2 Combination of parameters

This section analyzes the influence of combinations of values for the parameters *scope* and *number of weeks* for schedules with a tightness of 1. We only analyze schedules with a scope of '2 out of 4' and '2 out of 5', as Appendix C.3 shows that the number of remaining shifts is most sensitive for these values. Figures C.19 - C.20 show the corresponding graphs.

For schedules with a tightness of 1 and a scope of '2 out of 4', the number of remaining shifts is mainly high for schedules with a length of 6 or 10 weeks. For schedules with a tightness of 1 and a scope of '2 out of 5', the number of remaining shifts is mainly high for schedules with a length of 8, 10, or 13 weeks. The Weekend Planner is mainly sensitive for instances with a schedule length that is not a multiple of the scope period (i.e. 6 and 10 are not a multiple of 4; 8 and 13 are not a multiple of 5).

It is easily explained that a schedule length of 13 is not such a problem for instances with a scope of '2 out of 4': the maximum number of weekends an employee can work in this instance is $\lfloor \frac{2 \cdot 13}{4} \rfloor = 6$, the same number as if the schedule would have been 12 weeks, which is a multiple of 4. The schedule thus becomes a little easier: you have more weeks to schedule the same number of weekends. In the case of a schedule length of 6 or 10 weeks, the maximum number of weekends is respectively 3 and 5. This is one weekend more than for the schedule lengths that are a multiple of 4 (2 weekends for 4 weeks; 4 weekends for 8 weeks). The schedule is thus more complex, sometimes resulting in a high number of remaining shifts. Also very tight schedules with longer schedule lengths (8 and 10 weeks) that are a multiple of the scope form a problem sometimes. The results on the combinations of 8 weeks with a scope of '2 out of 4' and 10 weeks with a scope of '2 out of 5' show this.

An important remark on these findings is the probability that the described instances occur in practice. As said before, the Weekend Planner is mainly sensitive for cyclical schedules. A cyclical schedule in which the schedule length is not a multiple of the period of the scope will not very often happen in practice: rules are not easily checked and make a lot less sense. Users will select, if possible, a more logical schedule length.

5.5 Alternative algorithm designs

Select random shift in time

Tests have shown that excluding the rule to select the first shift in the time-line results in an improvement in comparison with the results found in Section 5.3, see Table 5.3 for a comparison. We would have expected this rule to deteriorate the results, because employees can have a history, which has more influence on the start of the planning period than towards the end of it. Obviously this expectation was not correct. The increased influence of randomness has a more positive influence on the results than selecting the first shift in the time-line.

Scenario	Average # remaining shifts per instance	% of instances with remaining shifts	Average relative deviation from optimal	Average absolute deviation from optimal	% of non-optimal instances
Base design of Weekend Planner	0.85	11%	3.8%	13.9	32.5%
Select random shift in time	0.78	9.75%	3.5%	13.2	30.5%
Regret-based random sampling	1.16	14%	10.9%	28.4	77%

Table 5.3: Comparison results Weekend Planner with alternative designs

Regret-based random sampling

We have performed tests on the Regret-based random sampling method as described in Section 4.4. We have tested various values for bias-factor γ and for the weights w_i for resource selection. Table D.1 in appendix D summarizes all performed tests. We have set the weight on the second and third shift selection rule to 0 for all tests, because the second rule does not make sense independently and the first alternative design option shows that excluding the third rule gives better results. We have tested 30 samples per instance, so 10 more than we have used in the tests on the Weekend Planner as randomness has more influence in Regret-based random sampling. Table 5.3 states the best achieved result with this method.

The results for Regret-based random sampling stay far behind on the results of the Weekend Planner (up to 200% deterioration on the average relative deviation from optimal). The increased influence of randomness does not have a positive contribution on the results of the Weekend Planner. The Weekend Scheduling Problem appears to be so complicated that too much randomness heavily deteriorates results. Increasing the number of samples can improve the results of Regret-based random sampling. As the gap between the results of Regret-based random sampling and the current Weekend Planner is very large, the number of samples will have to increase strongly. Even then, it is questionable how good the results will be seen the current gap and of course running time will also increase heavily then.

5.6 Comparison results for case study

This section summarizes the results for the three cases from the case study. It analyzes both the results for the various methods of planning in Harmony (Section 5.6.2) and the results for the Weekend Planner (Section 5.6.3). Both sections use performance indicators to analyze the results. Section 5.6.1 describes the used performance indicators.

5.6.1 Performance indicators

To be able to compare solution methods, we have defined performance indicators. This section describes the performance indicators for the scheduling of weekend shifts. Section 5.6.2 compares the solution methods from Section 2.3 for the cases from the case study using these performance indicators.

Remaining shifts

This is the difference between the demand of weekend shifts and the number of weekend shifts that is assigned to resources. In other words, the shifts that should be assigned to a resource, but that are still vacant. The unassigned shifts are measured in number of unassigned weekend shifts, i.e. in Harmony each weekend has two shifts that have to be assigned.

Equitable division of number of weekend shifts

An equitable division of number of weekends means that every resource works approximately the same number of weekends. This means that the difference between the number of weekends a resource works and the average workload per resource should be as low as possible. This indicator is quantified by using formula 3.1, which is described in Section 3.3. The three cases from the case study all use an equitable division of the absolute number of weekends (so they do not divide weekends in proportion to FTE).

Equitable division of shift types in weekend shifts

An equitable division of shift types in weekend shifts means that every resource works approximately the same number of weekend shifts of each shift type. So it is more equitable to have two employees working one weekend of day shifts and one weekend of night shifts than one employee

working two weekends of night shifts and one employee two weekends of day shifts. This indicator is quantified by using formula 3.2, which is described in Section 3.3.

Single weekend shifts

This is the number of weekends in which only one shift is assigned. For example a shift on Saturday followed by a free Sunday.

Consecutive weekend shifts of different shift types

This is the number of weekends in which more than one shift type is planned.

5.6.2 Results for cases with Harmony

This section discusses Harmony's results for the three cases from the case study. For each case the section first gives the user settings, followed by the results. Section 5.6.2.1 gives the general settings used in testing.

5.6.2.1 Settings

To run Harmony the user has to input the time the algorithm is allowed to run and the number of schedules the genetic algorithm starts with. In all tests performed in this section except for the optimization engine for the complete schedule, the algorithm is allowed to run for 1 hour (but will stop earlier if an optimal schedule is reached) and the genetic algorithm starts with 2 schedules. The test for the optimization engine for the complete schedule is allowed to run for 2 hours, as it is more complex than the planning for weekend shifts only.

The soft constraints users have defined are maintained in the tests. The section on each case summarizes the used constraints. The weights on the constraints have the same value in the tests as in the original cases.

scenario	Performance indicators				
	remaining shifts (week-end days)	equitable division number of shifts	equitable division shift types	single weekend shifts	weekends with different shift types
current schedule	0	0	6	0	0
optimization engine for complete schedule	14	4	26	12	1
shift sequence planner	8	8	22	0	0
optimization engine for weekend shifts	5	4	30	1	2
opt. eng. weekend shifts non-weekend rules off	0	0	30	0	0

Table 5.4: Results Calgary Health Region

5.6.2.2 Calgary Health Region

User settings

Tests are performed on a department with 42 employees, a four-week cyclical schedule with 3 different shift types on weekend demand. Employees are allowed to work a maximum of two out of four weekends and every employee is allowed to work only one or two shift types.

Calgary Health Region has defined the following soft constraints:

- No stand-alone weekend shifts
- Consecutive shifts have the same shift type.

Results

The results in Table 5.4 show that the current schedule, that Calgary Health Region has made manually, is clearly better than the schedules generated with Harmony. The second line in the results table shows the main reason for this research. When using the optimization engine for the complete schedule, the resulting schedule is far from optimal regarding the weekend shifts with 14 unassigned weekend shifts.

The three other approaches, which all need a second step to come to a complete schedule, give better results for the weekend shifts. The optimization engine for weekend shifts only with non-weekend rules turned off gives the best results. The penalty for a not equitable division of shift types is still quite high, but at least this method results in no unassigned shifts, which is the most important performance indicator.

This instance of Calgary Health Region scores high on tightness of the schedule: every employee has to work the maximum of two weekends to be able to assign all shifts. This means that if a planning method is able to result in zero remaining shifts, that the penalty for an equitable division of the number of shifts is automatically zero as well. This is the case for the optimization engine where non-weekend rules are turned off.

5.6.2.3 Belgian Police

User settings

Tests are performed on a department with 28 employees, an eight-week non-cyclical schedule with 5 different shift types on weekend demand. Employees are allowed to work a maximum of three out of four weekends.

The Belgian Police has defined the following soft constraints:

- Give every employee preferably one weekend of shifts of every shift type
- No stand-alone weekend shifts

The first rule emphasizes both an equitable division of the number of shifts and an equitable division of shift types.

Results

Table 5.5 shows the results for the Harmony tests on the case of the Belgian Police. For this case we only compare four results, because this user already plans using only weekend rules. So we left the scenario of using the optimization engine for weekend shifts with all rules on (including week rules) out of this analysis. For this case it also holds that the current schedule is obviously better than all other scenarios. Next, the scenario with the optimization engine with non-weekend rules turned off also is the only other scenario with zero remaining shifts. In this case the shift sequence planner does give a better result on single weekend shifts and weekends with different

scenario	Performance indicators				
	remaining shifts (week-end days)	equitable division number of shifts	equitable division shift types	single weekend shifts	weekends with different shift types
current schedule	0	126	120	0	0
shift sequence planner	14	172	189	0	0
optimization engine for complete schedule	1	272	230	57	28
opt. eng. weekend shifts non-weekend rules off	0	254	240	52	20

Table 5.5: Results Belgian Police

shift types, as their occurrence is impossible with the shift sequence planner. The Belgian Police currently uses the shift sequence planner followed by manual assignments and improvements.

5.6.2.4 Kennemer Gasthuis Haarlem

User settings

Tests are performed on a department with 28 employees, a four-week non-cyclical schedule with 4 different shift types on weekend demand. Employees are allowed to work a maximum of two out of four weekends.

The Kennemer Gasthuis Haarlem has defined the following soft constraint:

- Give every employee preferably one weekend of shifts of every shift type

This rule stimulates both an equitable division of the number of shifts and an equitable division of shift types.

It is remarkable that this user has not defined any rules on stand-alone weekend shifts or consecutive shifts of the same type, while they say to have a strong preference for these two criteria. The reason is that this user always uses the shift sequence planner for weekend shifts, which prevents the planner automatically from assigning stand-alone weekend shifts or consecutive shifts of different types.

scenario	Performance indicators				
	remaining shifts (week-end days)	equitable division number of shifts	equitable division shift types	single weekend shifts	weekends with different shift types
current schedule	0	46	94	0	0
shift sequence planner	0	52	88	0	0
optimization engine for complete schedule	7	48	102	13	15
optimization engine for weekend shifts	0	54	91	11	23
opt. eng. weekend shifts non-weekend rules off	0	50	90	10	24

Table 5.6: Results Kennemer Gasthuis Haarlem

Results

Table 5.6 shows the results for the Harmony tests on the case of the Kennemer Gasthuis Haarlem. In this case the results of the current schedule are approximately equal to those of the shift sequence planner. This is a logical result, because this user uses the shift sequence planner and hardly makes any manual adjustments. The optimization engine for the complete schedule is the only scenario that results in remaining shifts for this case. On the equitable division of the number of shifts and shift types all five scenarios have comparable scores.

5.6.2.5 Conclusions on case study results

The current possibilities in Harmony give diverse results for the three cases. The results of the shift sequence planner and the optimization engine with non-weekend rules turned off are closest to the current case schedules. We do have to make a remark on the number of remaining shifts when using the shift sequence planner for some cases. Remaining shifts are unacceptable for users, so this method is not always preferable either.

Although it appears to be a good idea to combine these two scenarios: use the shift sequence planner and turn off all non-weekend rules, this is not appropriate for all cases. In the case of the Belgian Police the non-weekend rules were turned off in all scenarios and the result for the shift sequence planner was not good at all.

An important disadvantage of turning off the non-weekend rules is the workload for users. Turning on and off rules is quite labor-intensive, whereas users strongly prefer to perform as least actions as possible. Another disadvantage of all shown scenarios is that most of them are worse than the current schedules, which leads to the conclusion that manual improvements seem to be inevitable with the current alternatives in Harmony.

We have discussed the disadvantages of the current possibilities in Harmony. As this section shows that the performance of Harmony differs per case, users should do some try-and-error for each case to see what scenario works best. The goal of the Weekend Planner is to take away the described issues. Section 5.6.3 shows the results for the cases using the Weekend Planner.

5.6.3 Comparison Harmony results with Weekend Planner results

This section gives the results of the Weekend Planner for the three cases and compares these results with Harmony's results. We use the best found scenario for the Weekend Planner: the alternative design without the *select first shift in time-line*-selection rule. We use the following algorithm settings: precision =1, samples = 20, local search iterations = 5. The Weekend Planner solves all three cases in 15-30 ms.

Table 5.7 shows the value of the performance indicators for the results of the Weekend Planner. For all three cases the Weekend Planner has zero remaining shifts. This is, at least for Calgary Health Region and the Belgian Police, an important improvement in comparison with Harmony's results.

For Calgary Health Region the result of the other four performance indicators is equal to their current, completely manually made, schedule. According to them, this is the optimal score achievable for this instance. We can thus conclude that the Weekend Planner performs very well for this case. The schedule of the Weekend Planner is 400% better than the best scenario in Harmony for the equitable division of shift types.

For the Belgian Police the result for the equitable division of the number of shifts is an improvement of more than 57% compared to their current schedule and more than 68% compared to Harmony's best result.. The result for the equitable division of shift types is around 23% worse than their current schedule and 22% better than Harmony's best result. As described earlier, we expect the result for this performance indicator to improve with a more extensive local search. We have to remark here that Harmony's best result on these two performance indicators resulted in 14 remaining shifts and is thus even worse than the percentages mentioned here.

For the Kennemer Gasthuis the results are slightly better than their current schedule and Har-

case	**Performance indicators**				
	remaining shifts (week-end days)	equitable division number of shifts	equitable division shift types	single weekend shifts	weekends with different shift types
Calgary Health Region	0	0	6	0	0
Belgian Police	0	54	148	0	0
Kennemer Gasthuis	0	39	78	0	0

Table 5.7: Weekend Planner results for case study

mony's best schedule (an improvement of 15% on the equitable division of the number of shifts and an improvement of 17% on the equitable division of shift types).

Overall, we conclude that the Weekend Planner performs very well for the three cases. The results on the performance indicators are equal to or better than the current schedules and are all better than Harmony's results. The improvement in terms of percentages varies per case from 15% to 68% for the equitable division of the number of shifts and from 17% to 400% for the equitable division of shift types.

Chapter 6

Conclusions and recommendations

This research is performed within ORTEC's Product Knowledge Center. The objective is to develop a method to improve the assignment of weekend shifts for ORTEC Harmony such that the total number of unassigned shifts and the penalties on key performance indicators are reduced. We have developed a construction heuristic, the Weekend Planner, to reach this objective and we have tested this solution in computational experiments. This chapter discusses the main conclusions and recommendations.

6.1 Conclusions

The current problems with the assignment of weekend shifts in Harmony's automatic plan functionalities follow from a case study on three real-life cases. These problems include schedules in which demand is not satisfied and non-equitable schedules in which employees do not work approximately the same number of shifts and shift types. In addition, the current alternatives in Harmony to create suitable schedules are very time-consuming.

Based on the problems found when using Harmony's automatic plan functionalities, we have modeled the Weekend Scheduling Problem (WSP) described in this research using a Quadratic Integer Programming formulation. The requirements for the solution for the WSP consist of solving the current problems in Harmony and it has to be possible to embed the solution in Harmony. Unsatisfied demand and equitability of a schedule are considered as key performance indicators. We have developed a Weekend Planner to solve the WSP. The Weekend Planner consists of a construction heuristic followed by a local search method. The construction heuristic iteratively selects a shift and a resource to assign that shift to. The selection of a shift is based on

three consecutive selection rules that assess the flexibility of a shift. The selection of a resource is based on three consecutive selection rules that on one hand assess the flexibility of a resource and on the other hand stimulate the equitable division of the number of shifts. The final local search method strives for a more equitable division of shift types. A random sampling method that iterates over the Weekend Planner creates multiple schedules and selects the best one.

We have tested the robustness of the Weekend Planner on 400 randomly generated instances, based on values often seen in practice. These tests show that the Weekend Planner is reasonably robust. It is mainly sensitive for cyclical schedules in which every employee has to work its maximum number of weekends and which have an irregular combination of schedule length and constraint on the number of weekends an employee is allowed to work. The sensitivity analysis also shows that the higher the number of samples, the better the resulting schedules. The analysis also shows that the higher the number of samples, the smaller the improvement is. We have used 20 samples per schedule, more samples only results in minor improvements.

Tests on three real-life cases from practice to compare the Weekend Planner with Harmony show that the Weekend Planner outperforms Harmony on all aspects. The Weekend Planner creates feasible schedules with no unfulfilled demand, and an improvement in the equality of the number of shifts (15% to 68%) and in the equality of shift types (17% to 400%). As the results of the Weekend Planner are according to users' requirements, no or little manual improvements have to be made.

Concluding, we provide a solution that improves the assignment of weekend shifts compared to the current results of ORTEC Harmony. The Weekend Planner is less time-consuming and thus a user-friendly method.

6.2 Recommendations

Based on the conclusions in the previous section, we give some recommendations for ORTEC in this section.

We recommend ORTEC to embed the Weekend Planner in the current structure of Harmony. The third shift selection rule should not be implemented as better results are obtained without this rule. We recommend to give Harmony users the choice whether they want weekend sequences to start on Fridays or on Saturdays, and to give them the possibility to define sequences themselves. When the user does not define sequences, the Weekend Planner should use the standard sequences as proposed in this research.

The current structure of the Shift Sequence Planner can be used for implementation of the

Weekend Planner. The construction heuristic of the Shift Sequence Planner then has to be replaced by the Weekend Planner. Defined shift sequences for weekend shifts should be scheduled with the Weekend Planner, other sequences are then scheduled by the current method.

We recommend to adjust the optimization engine of Harmony in such a way that it first schedules all weekend shifts before starting with the remaining shifts. As first step, the optimization engine should call the Weekend Planner. If the user has defined weekend sequences, it should use these. If not, it should use the standard sequences as defined in this research.

We recommend to use the local search method that is currently available in Harmony as post-processing instead of the local search method of the Weekend Planner. Harmony's local search is more extensive than the local search we have used in the Weekend Planner and it also tries to minimize the number of remaining shifts, which our local search does not do.

Bibliography

Burke, E., P. de Causmaecker, G. van den Berghe, H. van Landeghem, 'The State of the Art of Nurse Rostering', *Journal of Scheduling 7*, pp. 441-499, 2004.

Fijn van Draat, L., G. Post, B. Veltman, W. Winkelhuijzen, 'Harmonious personnel scheduling', *Medium Econometrische Toepassingen 14,* pp 4-7, 2006.

Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, 1989.

Kolisch, R. and A. Drexl, 'Adaptive Search for Solving Hard Project Scheduling Problems', *Naval Research Logistics 43,* pp 23-40, 1996.

Van der Put, M., *Master thesis,* Delft University of Technology, 2005.

Appendix A

Preliminary study - Calgary Health Region

A.1 Introduction to Calgary Health Region

The Calgary Health Region is one of the largest fully-integrated, publicly-funded health care systems in Canada. Serving a population of over 1.2 million people, the Region is home to some of the fastest growing communities in the country. The Calgary Health Region (CHR) serves the city of Calgary as well as a group of smaller communities - a total area the size of Switzerland. 29,000 employees and 2,300 physicians provide service in over 100 locations, including 12 hospitals, 4 comprehensive health centers, 41 care centers, and a variety of community and continuing care sites.

In March 2008, CHR purchased Harmony from ORTEC. They require Harmony to be able to automatically create cyclical rosters, for which ORTEC needed to make adjustments in Harmony, as this was not yet available in the software. The adjustments to be made are split up in different phases. In phase 1, delivered in the first week of August 2008, the functionality of the optimization engine for cyclical schedules is integrated in the software. In phase 2, delivered in November 2008, more functionalities that are necessary for CHR will be integrated. This preliminary research has taken place in September-October 2008. Although phase 2 was not delivered yet, Harmony should already have enough functionalities to create correct cyclical schedules. A cyclical schedule is also called a rotation, the personal schedule of an employee is called a rotation line.

The Rotation Management team of CHR has tested the new software as delivered in phase 1. Except for the functionalities to be delivered in phase 2, working with the phase 1 version of

Harmony did not completely satisfy the needs of CHR. It was not clear yet what was exactly their problem. From the 23rd of September 2008 until the 24th of October 2008, I have visited the Rotation Management Team and worked with them to define their problem and find a solution for it.

The Rotation Management Team of CHR makes rotations for departments of hospitals in the Health Region that request a new cyclical schedule. The Rotation Management Team creates a general cyclical schedule, which will be converted to a daily schedule by the department schedulers. Lead times tend to be quite long at this moment. From start to end it takes approximately 3 months to create a new cyclical schedule. Waiting time for a new cyclical schedule is four to eight months. The Rotation Management Team wants to shorten lead times. Therefore, they have purchased Harmony to support the consultants with creating cyclical schedules, which should shorten cycle time and thereby also waiting time and thus total lead time.

This study analyzes the working process of the Rotation Management Team (Section A.2), the control of Harmony (Section A.3) and the performance of Harmony at CHR (Section A.4). Each section describes the current status, the changes we have made and gives a conclusion on the subject.

A.2 Process

A.2.1 Current process

The schedules of employees of CHR are cyclical. Depending on the unit they work 4, 8, or 12 week cyclical schedules. Every employee has its own unique line in the schedule (called a rotation line), so CHR in general does not use block rotations. With a block rotation we mean a cyclical schedule in which every employee works the same line, but every employee starts in a different week of the line. The cyclical schedule will be used until the manager feels the need to change the cyclical schedule, which can for example be forced by complaints of employees. A change of employees or a change in union rules does not necessarily have to be a reason to change the cyclical schedule. A vacant position will be posted including the rotation line to be worked. A change in union rules might result in cyclical schedules not being contract-compliant anymore, but the cyclical schedule only has to be adapted when employees request it. They might just as well like their rotation line, although it is not contract-compliant anymore.

Cyclical schedules are generally made by the unit managers. When they have a cyclical schedule they want to use, they send it to their scheduler, who inputs the cyclical schedule in ESP, the software CHR currently uses for daily planning. In 2006 CHR set up a Rotation Management

Team that supports managers with their cyclical schedules and the scheduling process. The Rotation Management Team works on a request-basis. A manager of a unit within CHR can request a consultation of the Rotation Management Team when they want to review or renew the cyclical schedule of their unit. The Rotation Management Team consists of five rotation consultants, a rotation analyst and a rotation assistant. The rotation analyst and the rotation assistant support the consultants, who are responsible for making the cyclical schedules and are the direct contact persons for the unit managers. The next paragraph discusses the consultation process in more detail.

A consultation starts when a manager submits a request form. The manager provides information about the current cyclical schedule, possible grievances, vacancies and expected time line when filing a request. One of the consultants then gathers unit information from the unit scheduler in the scheduling office, does a preliminary analysis of the current cyclical schedule and plans an initial meeting with the unit manager. In this initial meeting the manager provides more detailed information on the unit, for example the occupancy requirements, and challenges. The unit manager and the consultant also discuss the current cyclical schedule, what they both do and do not like about it. The consultant reviews the schedule mainly from a contract perspective, the manager mainly from an employee and unit perspective. They set a time line for the process of developing a new cyclical schedule, discuss the next steps, and discuss whether the manager wants the consultant to ask the unit employees for their preferences in a new schedule or not. In most cases the manager has to gather additional information after the first meeting. Meanwhile the rotation analyst gathers additional information on budget, actual needed staff based on workload data, etc. The consultant sets up a requirements summary, presents the project and its purposes to the employees of the unit and, where applicable, hands out the employee survey. Manager and consultant then have a review meeting to finalize the data, after which the consultant builds a new cyclical schedule, which takes approximately 1-5 days, depending on the size and complexity of the unit. More discussions with the manager follow to improve and finalize the cyclical schedule. In practice this results in an endless ongoing iterative process of building a cyclical schedule, managers changing their preferences, consultants building again, managers coming up with new preferences, etc.

In general, rotation lines are built for certain employees, so employee requests are applied to a specific line in the cyclical schedule. However, when a cyclical schedule is ready, the employees have to approve their line. If only one of the employees does not approve his/her rotation line, all employees have to pick a line by seniority. Seniority is based on the number of years the employee has been working for the same union. Finally, a by all employees approved cyclical schedule is sent to the scheduler, who inputs the cyclical schedule in ESP.

Currently, the consultants make the cyclical schedules using an Excel template. In the template various calculations are shown. Union rules have to be checked manually. When consultants start making a cyclical schedule, they first plot the weekends, followed by the full time lines and then they plot the remaining shifts. Obviously, this is a time consuming process. Harmony can reduce the time spent on cyclical schedules drastically by checking the rules automatically and using the optimization engine.

A.2.2 Conclusion

The iterative process of managers defining preferences and consultants building a cyclical schedule is not suitable in combination with the use of Harmony, as this will increase the workload considerably for the rotation analyst, who has to input unit preferences. A possible way to improve processes in combination with Harmony is to first analyze what information the consultants need from the managers and what subjects managers need to decide on. The consultants can do this by keeping track of the subjects that managers bring up during the iterative process. The consultants should make a list of all information they need to gather from managers and only start building the cyclical schedule once they have all necessary information. They should also make clear to managers that after a certain deadline, preferences cannot be changed anymore. This will mean that a consultant has multiple meetings with a manager before starting to build the cyclical schedule, instead of starting to build after the first meeting as it is right now. Building the cyclical schedule will be done later in the process compared to the current situation, but will be considerably faster as it is not an iterative process anymore.

When implementing the new software, CHR obviously has to review their roles and has to decide on responsibilities in the new process. Based on this study, the advice to CHR is to implement the following roles and responsibilities. Before the first meeting between a consultant and an unit manager, the rotation assistant creates the unit in Harmony, inputs all employees and the corresponding data, the current shifts, occupancy requirements, and the current cyclical schedule. The consultant then takes information about the current cyclical schedule to the unit manager and discusses it. When changes in preferences, shifts, baseline and employee data have become clear, the rotation assistant updates the unit in Harmony. After discussing the requirements with the consultant, the rotation analyst creates work agreements for the unit and sets up scheduling criteria, including unit preferences. The consultant is responsible for making a new cyclical schedule.

A.3 Control

A.3.1 Harmony changes

As Harmony did not include an optimization engine to create cyclical schedules, ORTEC has made changes to Harmony for CHR. The main new functionality in phase 1 was the automatic planner for cyclical schedules, delivered in July 2008. Another problem CHR came across is that the minimum demand of a unit mostly does not contain enough shifts to cover the employees FTE. Therefore, CHR needs to add shifts over baseline to complete employees FTE's. ORTEC will be able to include functionalities for these additional requirements in phase 2. More functionalities that this study has shown CHR needs in the cyclical schedule, such as visually showing certain data and calculations and using daily requirements for combinations of 8 and 12 hour shifts in one unit (as there is overlap between those shifts), will be addressed in phase 3 or later.

A.3.2 Definition of scheduling requirements in the software

The scheduling requirements in CHR consist of union rules, work agreements and general scheduling criteria. Employees of CHR belong to four different unions: UNA, HSAA, AUPE GSS, AUPE AUX. It depends on the position to which union an employee belongs. Most employees belong to either UNA (Registered Nurses) or AUPE AUX (Licensed Practical Nurses). Different labor rules are set up for each union, depending on the number of hours per shift (8, 10 or 12), so i.e. UNA 8, UNA 12, etc. Work agreements are linked to individual employees and contain information about the shift pattern to be worked, the preferred number of shifts in a row (depending on FTE), and the minimum number of days to be worked (depending on FTE and union rules, i.e. 40% day shifts for UNA employees). Examples of shift patterns are: only evenings, only nights, a day/evening pattern or a day/night pattern.

The setup of the above described work agreements was not done efficiently and the setup of labor rules was not a correct translation of CHR processes and rules to the software. To improve both of these areas, we have changed the setup of the work agreements to be more efficient and robust. To every employee now several work agreements are added. The general scheduling criteria are used unit-wide for preferences, such as no stand-alone shifts, working complete weekends (no parts of it), etc. Together with one of the CHR employees I have corrected and tested the labor rules, which has resulted in a correct translation of the rules in Harmony.

A.3.3 Conclusion

The members of the Rotation Management Team have significantly improved their knowledge of the use of Harmony during the month we have worked together. Their knowledge currently is at a sufficient level to start working with Harmony. All rules and regulations are set up correct in Harmony.

A.4 Performance

A.4.1 Quality of cyclical schedule

While testing the software in Calgary, the software did not give a satisfying result. We have tested on a unit that one of the consultants already made a new cyclical schedule for, manually. Even after running for a night, Harmony was not able to come up with a result close to the manually made cyclical schedule. The result was a higher number of unassigned shifts (9 in Harmony compared to 1 in the manually made cyclical schedule) and a lower quality of the cyclical schedule (more penalties on criteria in the Harmony cyclical schedule). We found two main bottlenecks in Harmony:

- not being able to create a good cyclical schedule for the full-time lines
- not being able to assign all weekend shifts

In the manually made cyclical schedule it is possible to achieve a good line for full time employees and to assign all weekend shifts. To overcome these two main problems for CHR, I have come up with a short-term solution. This solution consists of 3 consecutive steps. When performing these steps, the cyclical schedule Harmony finds is considerably better than just using the optimization engine.

1. Input full time lines manually.
2. As the weekends are quite tight in most cases, it is best to optimize them separately. So run the optimization engine for the weekend shifts. Weekends then have to be tweaked manually, to divide them equally over all employees and to assign weekends Harmony could not assign yet. In most cases you can find those assignments manually.
3. Use the optimization engine to plan all remaining unassigned shifts. The consultants will then start tweaking to increase the quality of the schedule, as there always are unit specific preferences Harmony cannot deal with.

Although this 3-steps approach improves the quality of the cyclical schedule and reduces the time spent on making a schedule, the preferred situation of course would be that Harmony could do this automatically. Besides that, we still encounter some problems when using the planners in the cyclical schedule in Harmony. The shift sequence planner only plans the shift sequences and does not optimize them. This results in an inefficient assignment of shift sequences, what makes the shift sequence planner at this moment a tool with little value for the cyclical schedule. Another important issue is optimizing the weekends using the optimization engine in step 2. Planning the weekend shifts results in a reasonable assignment, although Harmony is not able to plan all weekend shifts, whereas this is possible by manual assignment.

A.4.2 Conclusion

As described in Section A.4.1 planning the weekends and full time lines are difficulties in Harmony. It is quite intuitive that Harmony cannot find the full time lines, as only 2 or 3 lines are contract compliant whereas a huge number of lines exist that Harmony tries to fill in. The chance that Harmony finds exactly one of the few contract compliant lines is quite small. Inputting the full time lines manually thus seems to be most efficient. For the assignment of weekend shifts it is quite important to find a solution in Harmony. The impact of having a bad weekend assignment is enormous, for several reasons both in- and outside of Harmony.

Harmony related reasons are that weekends are involved in many rules, which makes the assignment of a vacant weekend shift complicated once all other shifts are planned. Weekends are quite tight in most cases. What we mean by 'tight' is that there are just as much weekends to be assigned as total number of weekends employees are allowed to work. This means that only a few possible assignments for the weekends exist. The need for planning weekends first, separately from all other shifts, arises from the tightness of weekends.

In meetings with unit managers in Calgary we also found several non-Harmony related reasons for the large impact of a bad weekend assignment. First, having vacant shifts on weekends means that you have to find employees to cover those shifts. An employee rather covers a regular day shift than a weekend shift and on a weekend you have to pay irregularity allowance, which does not happen on regular days. Covering weekend shifts is thus more complicated and costly than regular shifts, so managers rather not have vacant shifts on the weekends. Second, weekend shifts have a large impact on the private life of employees. Weekends are the moments most other people do not work, children do not go to school, etc. Most employees thus want to have at least some weekends off to spend time with family or friends. This importance of weekends also calls for a fair and equitable division of weekend assignments among employees, so not one employee

working three weekends and somebody else working just one weekend.

Considering the importance of having a good weekend assignment combined with the inability of Harmony to deal with weekends in a satisfying way leads to the conclusion that we have to improve Harmony on the part of weekend assignment.

Appendix B

Experimental results

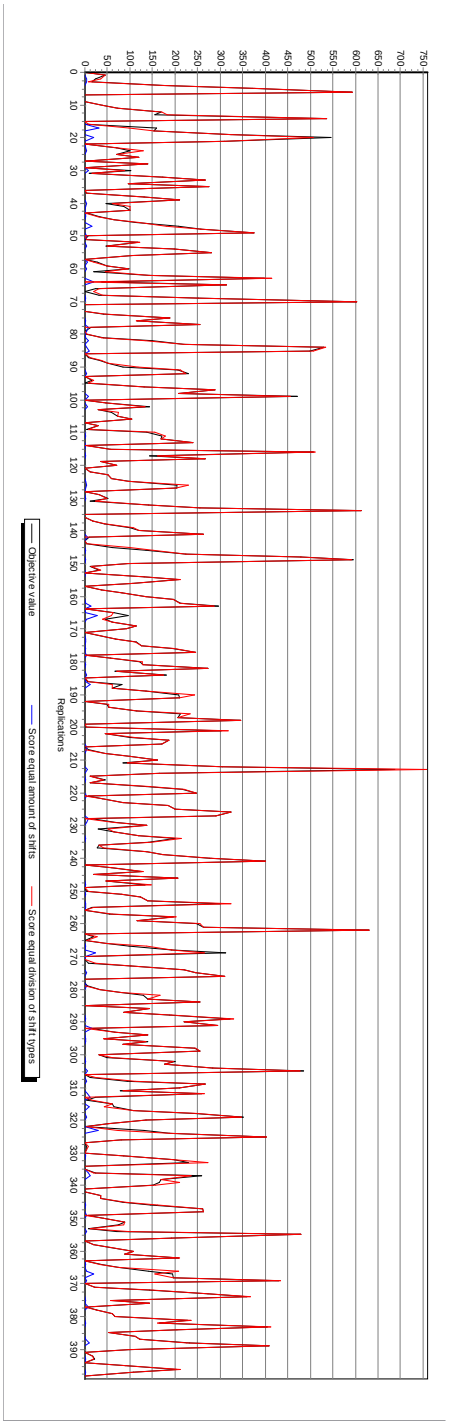

Figure B.1: Total penalty for random instances, specified in penalty for equal number of shifts and equal division of shift types

Appendix C

Sensitivity analysis

C.1 Instance size

This section analyzes the influence of the instance size on the results of the Weekend Planner. The instance size is determined by the number of employees and the number of weeks (length of the schedule). See Figures C.1 - C.3 for the graphs on the number of employees and Figures C.4 - C.6 for the graphs on the number of weeks.

Number of employees

The number of employees does not have an obvious influence on the remaining shifts or on the deviation from the optimal value. The number of remaining shifts and the absolute deviation from optimal slightly increase with a higher number of employees. As instances with more employees have more shifts to schedule, this is a logical increase.

Number of weeks

The number of weeks appears to have more influence on results than the number of employees. The higher the number of weeks, the higher the deviation from optimal (both relative as well as

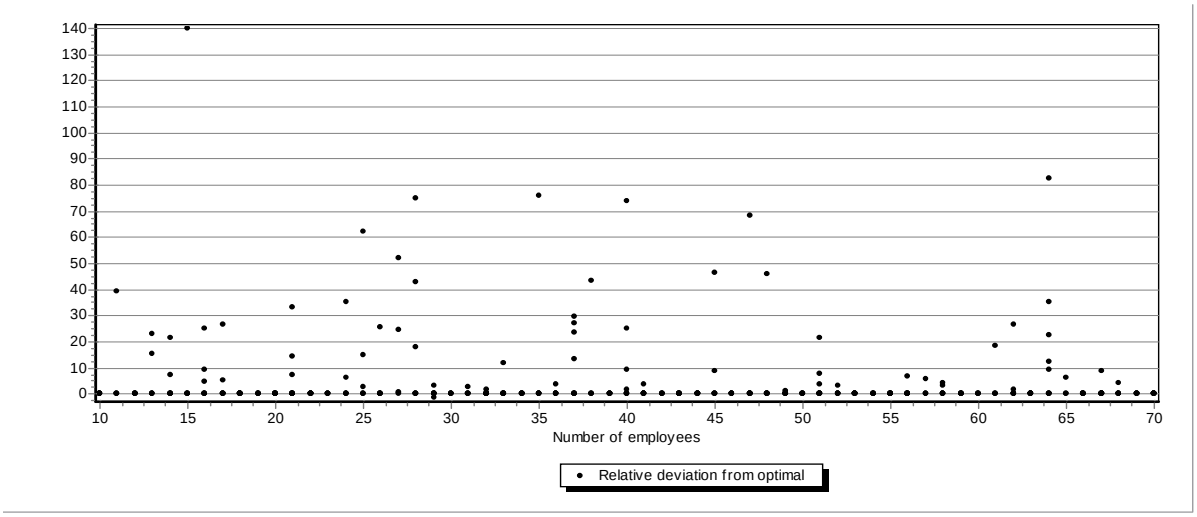

Figure C.1: Relation between number of employees and relative deviation from optimal solution

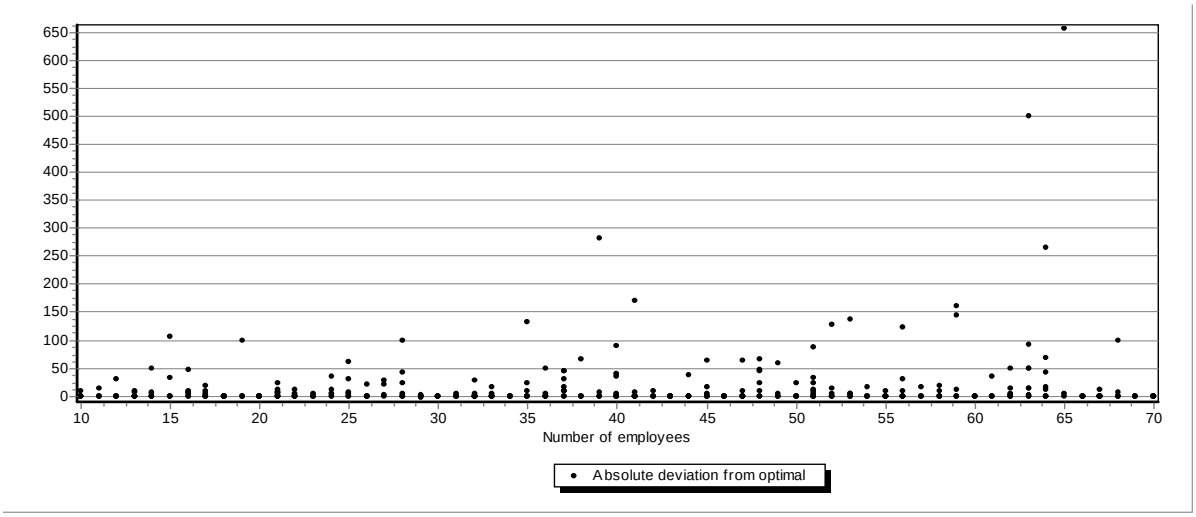

Figure C.2: Relation between number of employees and absolute deviation from optimal solution

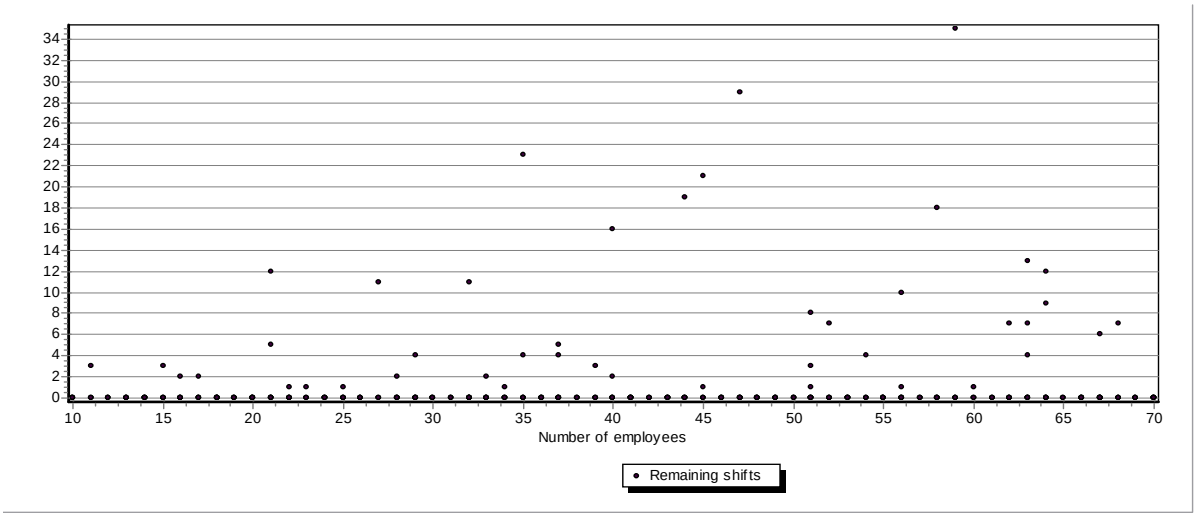

Figure C.3: Relation between number of employees and remaining shifts

absolute). The remaining shifts appear to be equal sensitive for instances of 6, 8, and 10 weeks. Instances of 13 weeks appear to have a better result on remaining shifts. The schedule lengths of 6, 8, and 10 weeks have relatively high remaining shifts, but no clear correlation is visible. That is why we assume that there is another parameter that influences the number of remaining shifts in combination with certain schedule lengths. The next sections analyze possible other parameters and combinations of parameters

C.2 Availability

This section analyzes the influence of the availability of employees on the results of the Weekend Planner. See Figures C.7 - C.9 for the corresponding graphs.

We expect that the lower the availability of employees, the more difficult the instance is. For the deviation from optimal this expectation appears to be true. The lower the availability, the higher the deviation from optimal is. On the contrary, the availability does not have an obvious influence on the number of remaining shifts.

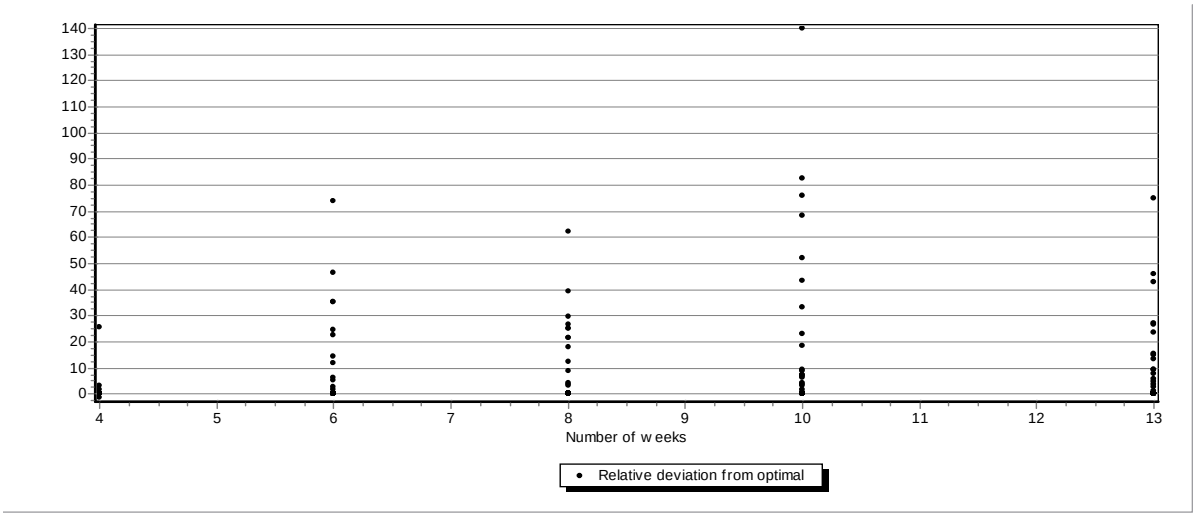

Figure C.4: Relation between number of weeks and relative deviation from optimal solution

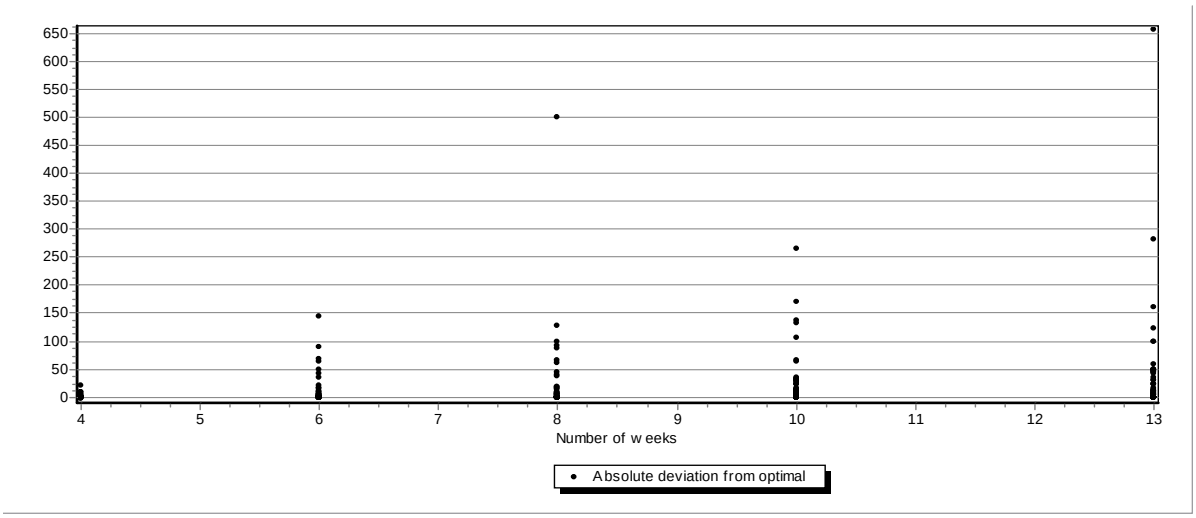

Figure C.5: Relation between number of weeks and absolute deviation from optimal solution

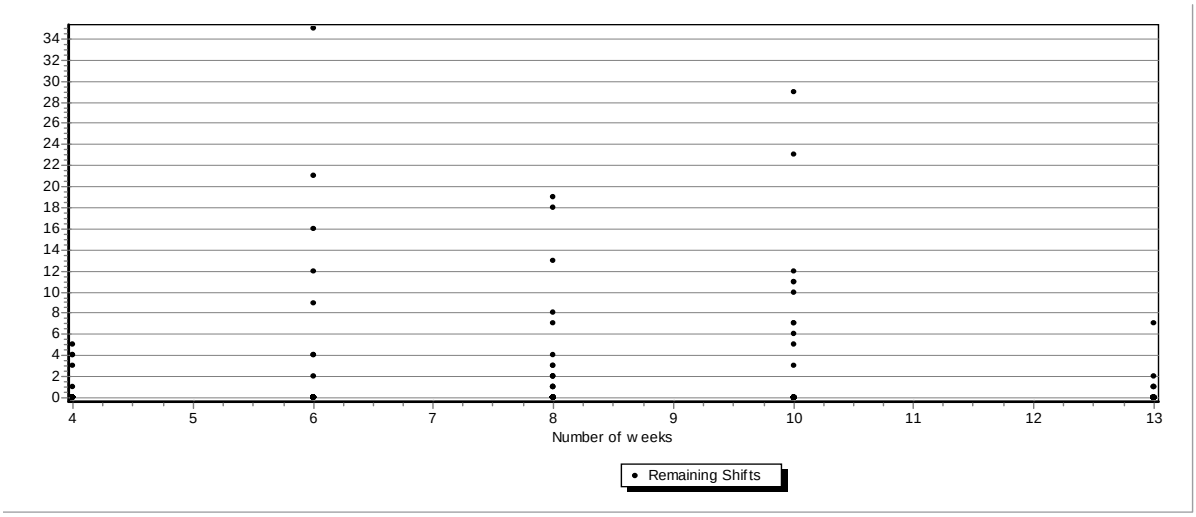

Figure C.6: Relation between number of weeks and remaining shifts

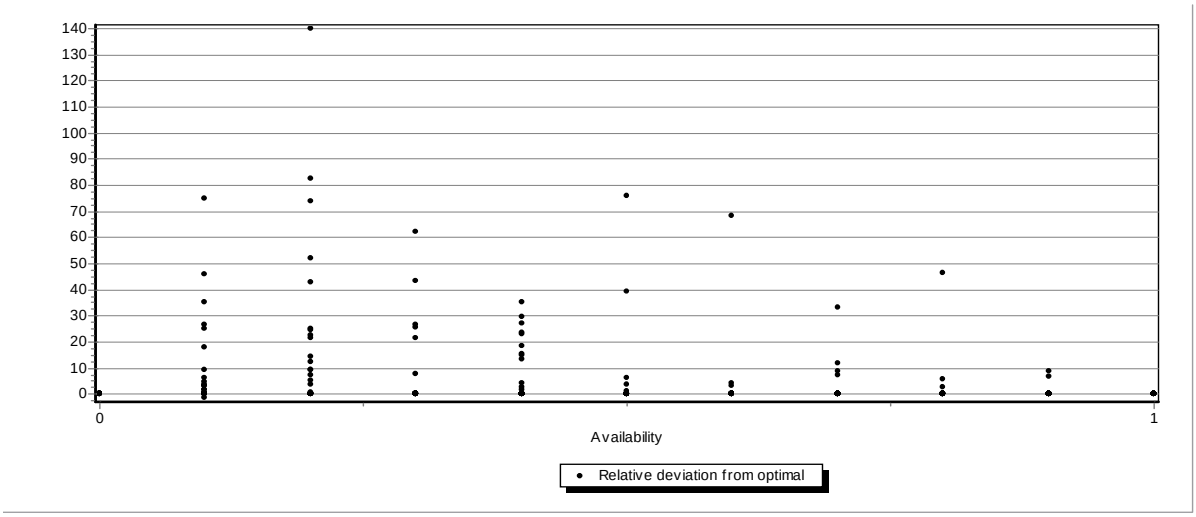

Figure C.7: Relation between availability and relative deviation from optimal solution

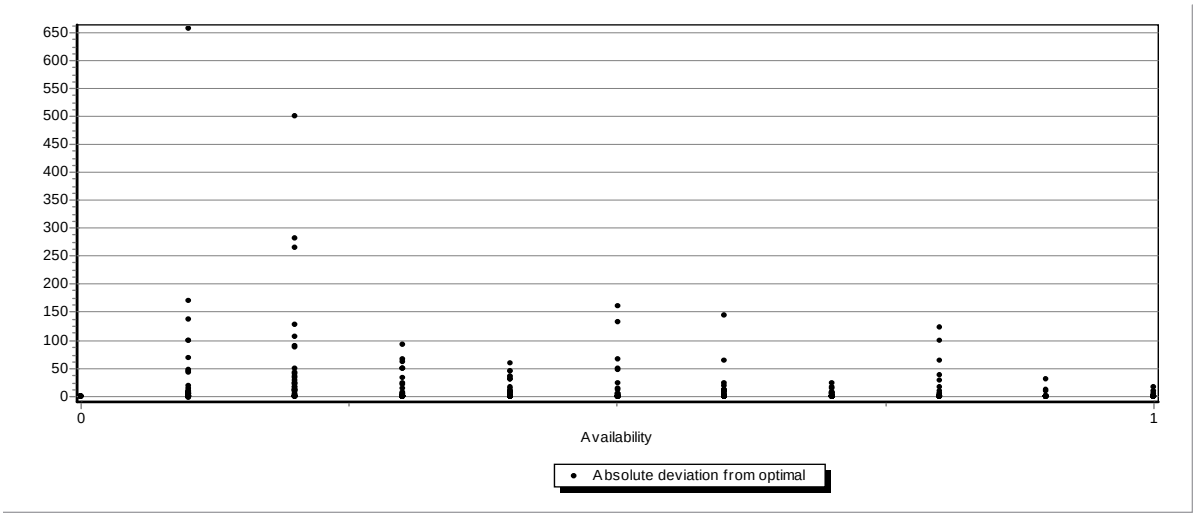

Figure C.8: Relation between availability and absolute deviation from optimal solution

C.3 Scope

This section analyzes the influence of the scope for the maximum number of allowed weekends to work on the results of the Weekend Planner. Figures C.10 - C.12 show the corresponding graphs.

The scope does not have an obvious influence on the deviation from optimal, all scopes have comparable results, where 2 out of 4 weekends and 3 out of 4 weekends have slightly worse results. The graphs show a more obvious correlation for the number of remaining shifts. A scope of 2 out of 4 weekends and 2 out of 5 weekends result in the highest numbers of remaining shifts.

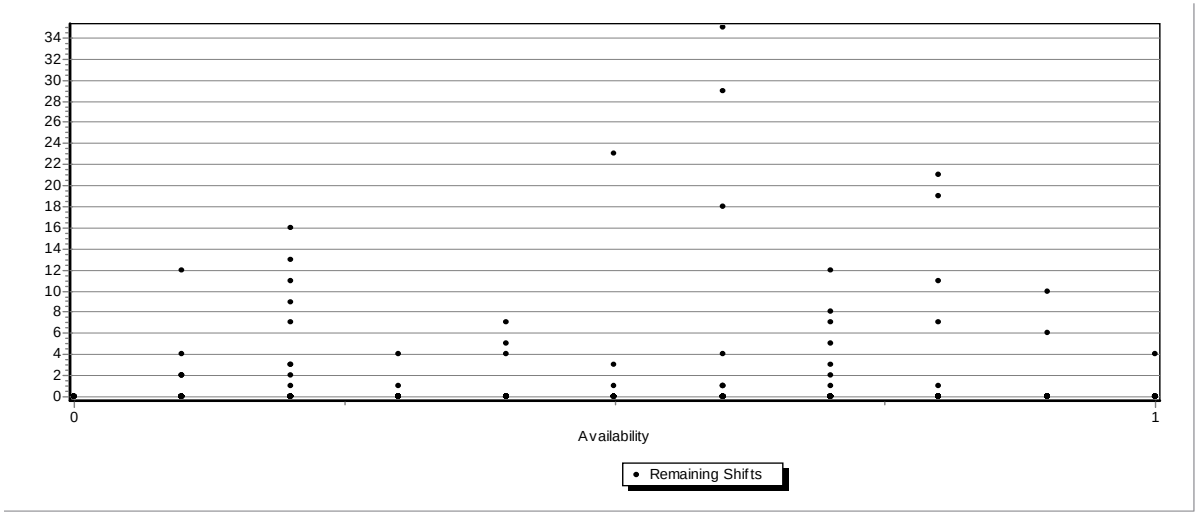

Figure C.9: Relation between availability and remaining shifts

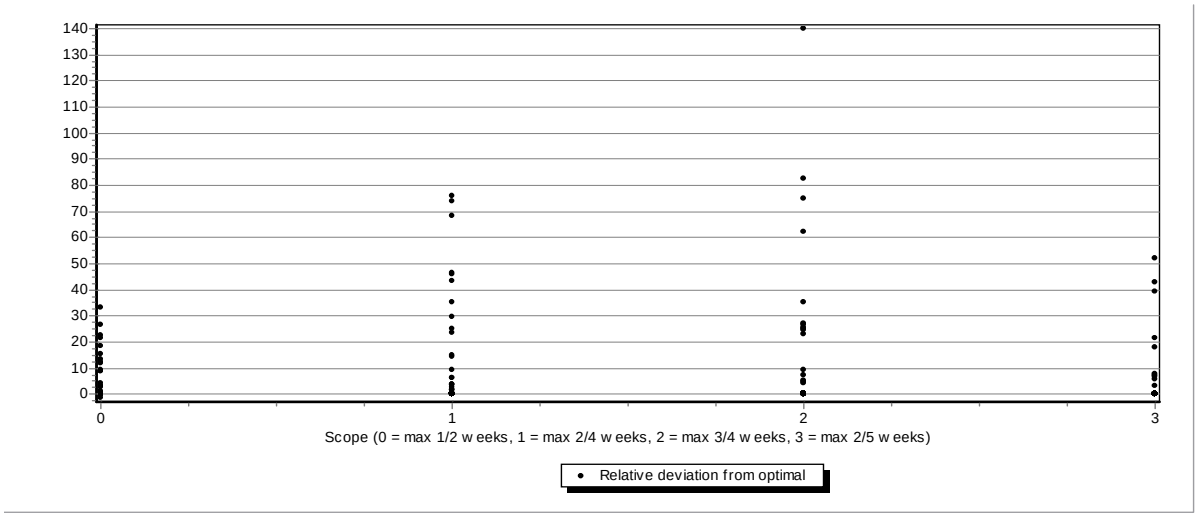

Figure C.10: Relation between scope and relative deviation from optimal solution

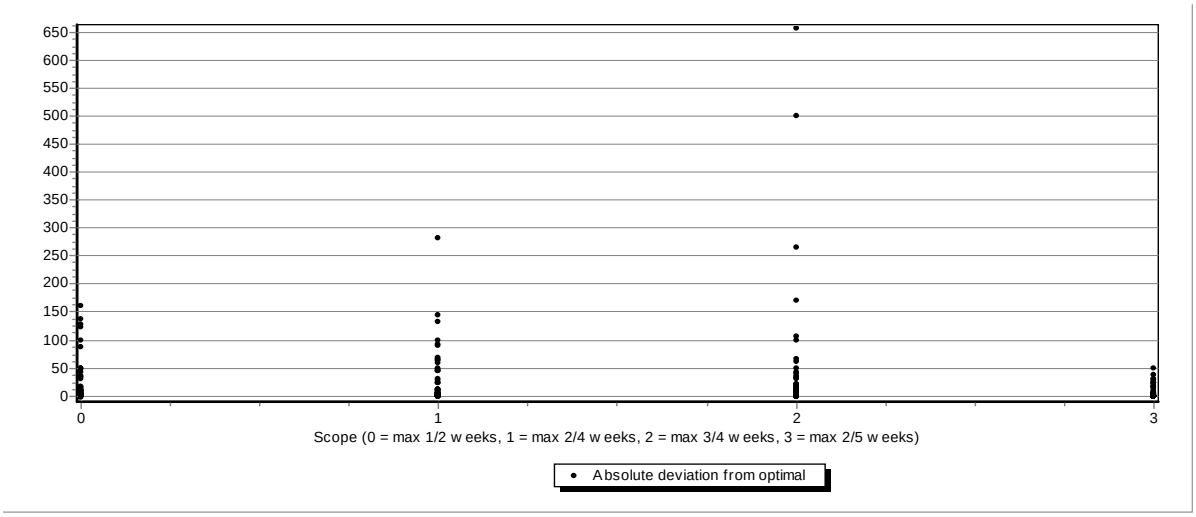

Figure C.11: Relation between scope and absolute deviation from optimal solution

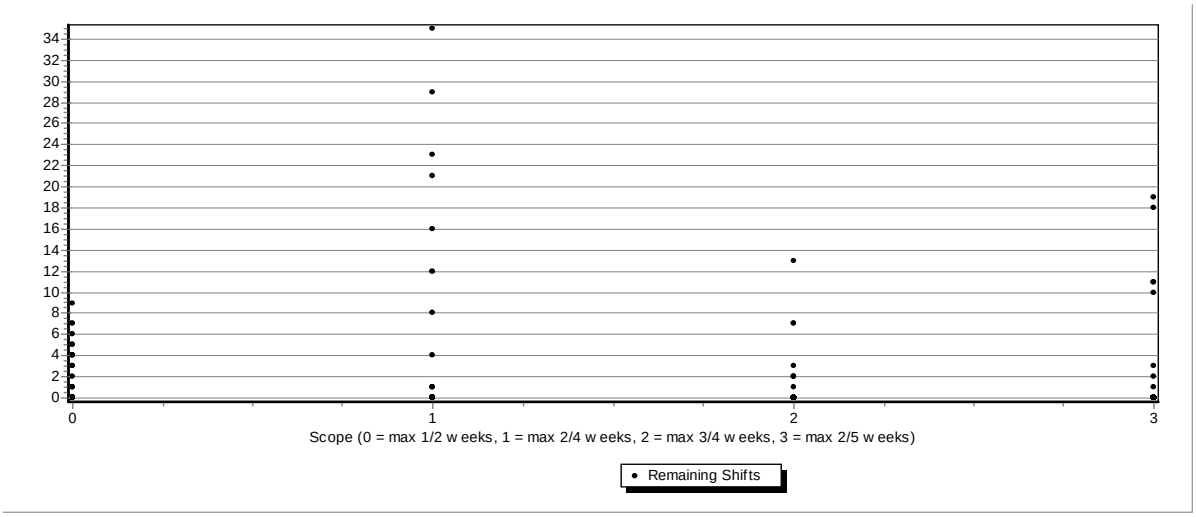

Figure C.12: Relation between scope and remaining shifts

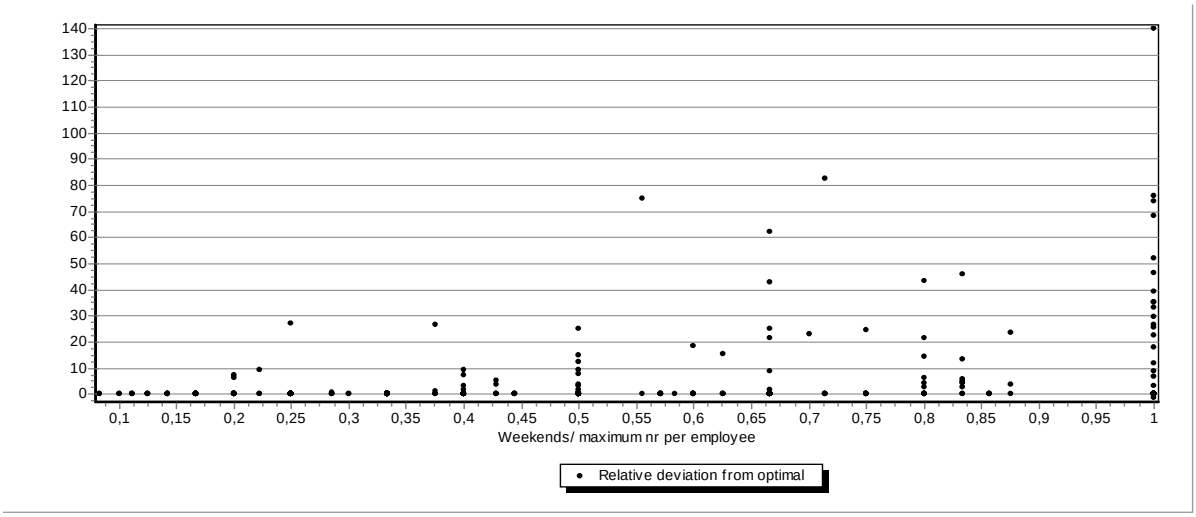

Figure C.13: Relation between tightness of schedule and relative deviation from optimal solution

C.4 Tightness of schedule

This section analyzes the influence of the tightness of the schedule on the results of the Weekend Planner. The tightness of the schedule is calculated as the average number of weekends per employee divided by the maximum number of weekends employees are allowed to work. This maximum is calculated as: $\frac{(number\ of\ weeks) \cdot (maximum\ number\ of\ weekends\ per\ period)}{(period\ for\ which\ maximum\ is\ calculated)}$. Figures C.13 - C.15 show the corresponding graphs.

The tightness of the schedule has an obvious influence on both the deviation from optimal as well as on the number of remaining shifts. The deviation from optimal increases approximately linearly with an increase in tightness, where schedules with a maximum tightness of 1 also have the largest deviation. The number of remaining shifts increases only slightly with an increase in tightness, but the number of remaining shifts for schedules with a tightness of 1 is extremely high. Almost all instances with unacceptable high remaining shifts, have a tightness of 1.

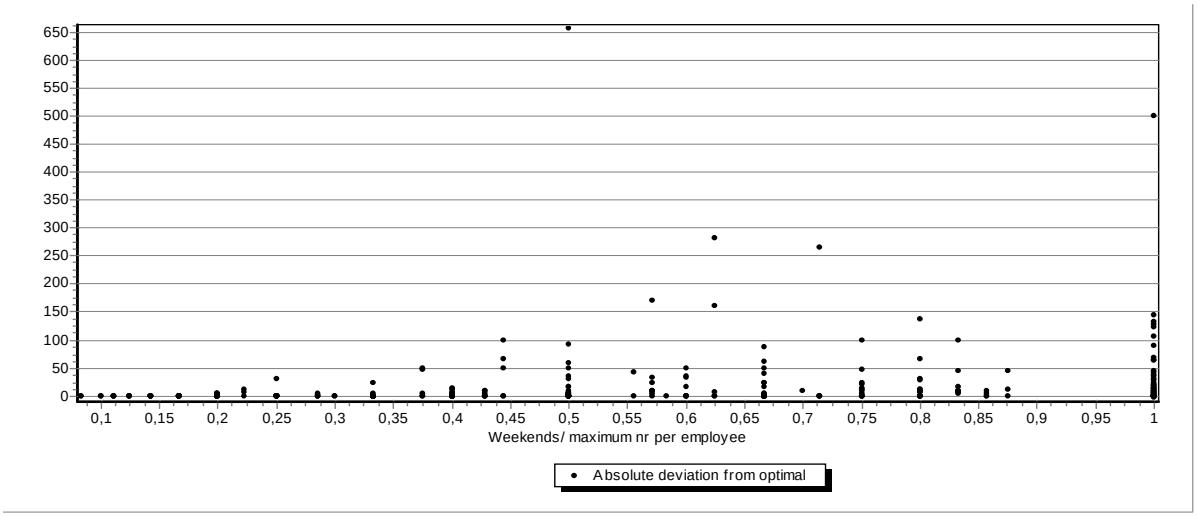

Figure C.14: Relation between tightness of schedule and absolute deviation from optimal solution

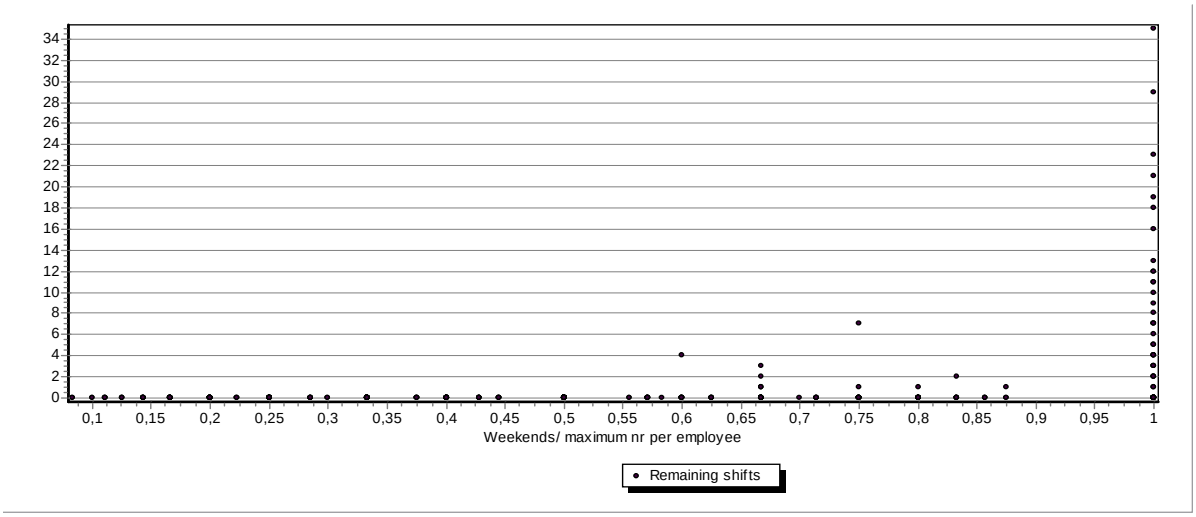

Figure C.15: Relation between tightness of schedule and remaining shifts

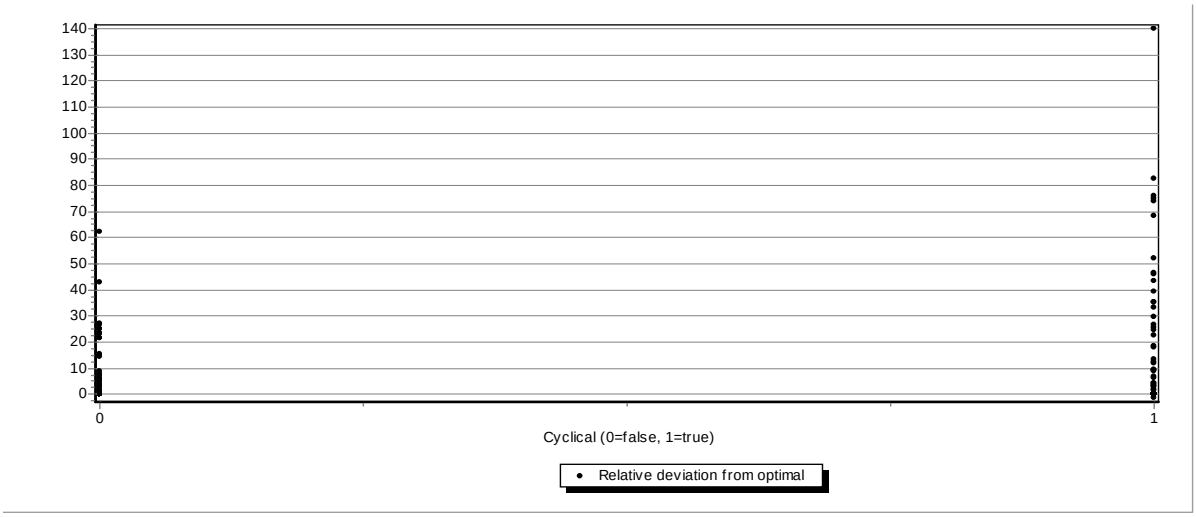

Figure C.16: Relation between cyclical and relative deviation from optimal solution

C.5 Cyclical

This section analyzes whether a schedule being cyclical or not has an influence on the deviation from optimal and on the remaining shifts. Figures C.16 - C.18 show the corresponding graphs.

For both cyclical and non-cyclical schedules the deviation from optimal is comparable, with a slightly higher deviation for cyclical schedules. This is explained by cyclical schedules being more complex as constraints for the end of the schedule have to be checked with the beginning of the schedule. For non-cyclical schedules this is not the case, which makes non-cyclical instances a little easier to solve. The difference in complexity between cyclical and non-cyclical schedules is best shown on the side of the remaining shifts. The number of remaining shifts is clearly higher for cyclical schedules than for non-cyclical schedules.

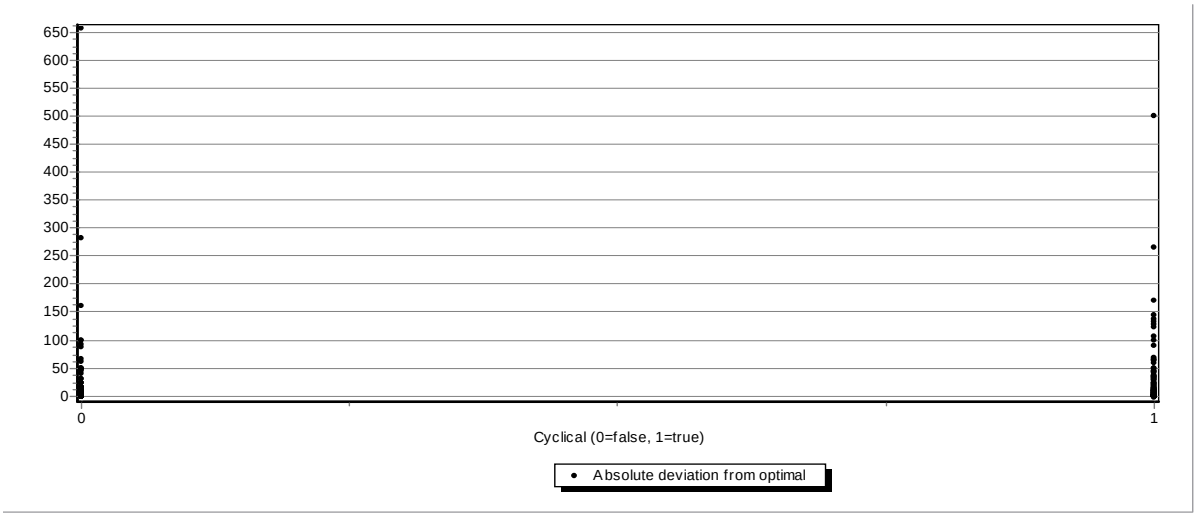

Figure C.17: Relation between cyclical and absolute deviation from optimal solution

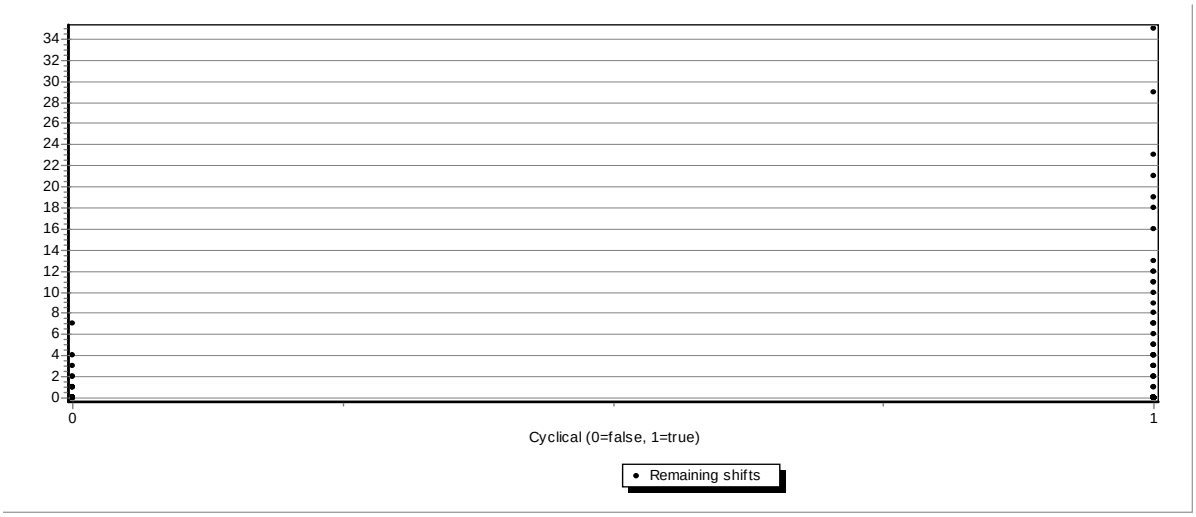

Figure C.18: Relation between cyclical and remaining shifts

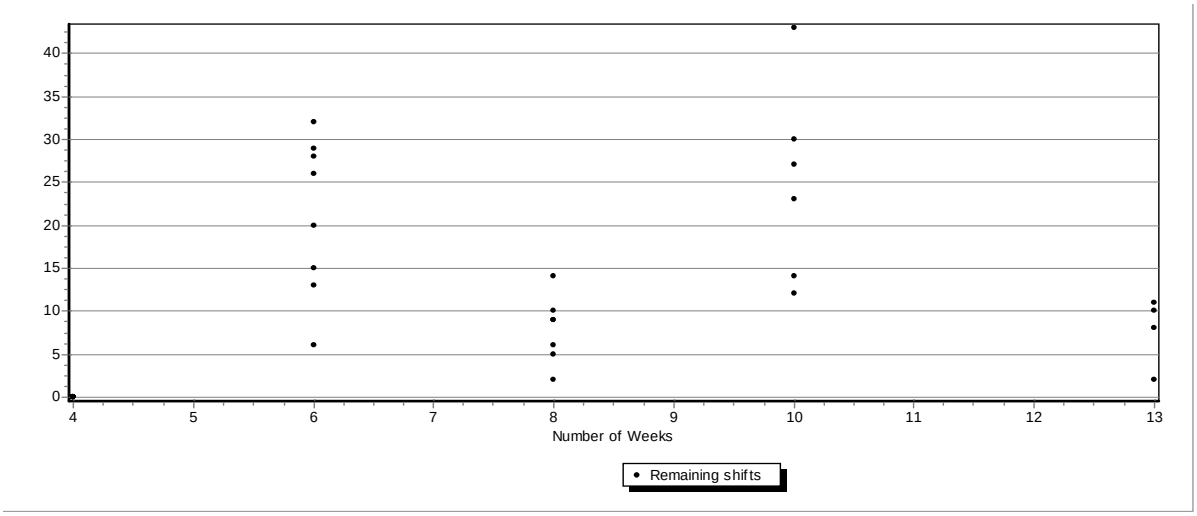

Figure C.19: The number of remaining shifts per schedule length for schedules with a tightness of 1 and a scope of working a maximum of 2 out of 4 weekends

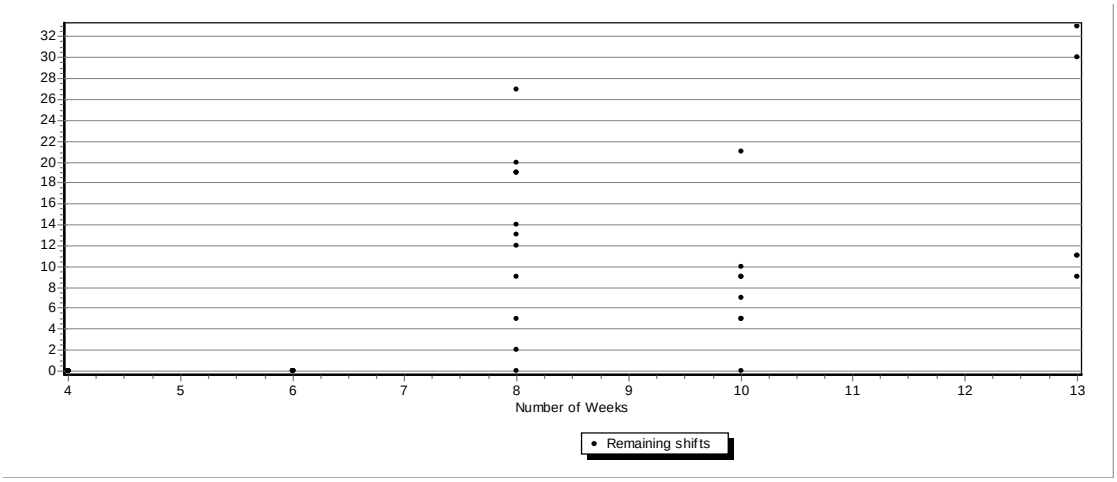

Figure C.20: The number of remaining shifts per schedule length for schedules with a tightness of 1 and a scope of working a maximum of 2 out of 5 weekends

Appendix D

Regret-based random sampling

Shift			Resource								
w_1	w_2	w_3	w_1	w_2	w_3	γ	Average # remaining shifts per instance	% of instances with remaining shifts	Average relative deviation from optimal	Average absolute deviation from optimal	% of non-optimal instances
1	0	0	0.5	0,5	0	2	1.29	15%	14.9%	35.5	81.25%
1	0	0	0.5	0.4	0.1	2	1.455	17%	12.4%	30.8	78%
1	0	0	0.5	0.3	0.2	2	1.545	16.5%	11.6%	29.2	78%
1	0	0	0.5	0.3	0.2	3	1.325	16.5%	11.1%	28	79%
1	0	0	0.5	0.3	0.2	4	1.155	14%	10.9%	28.4	77%
1	0	0	0.6	0.3	0.1	2	1.52	19.5%	11.1%	28.6	80%
1	0	0	0.7	0.2	0.1	2	1.6	18.5%	9.9%	25.9	78.5%

Table D.1: Results for Regret-based random sampling in the Weekend Planner