Titre: Title:	Optimization of heterogeneous employee scheduling problems
Auteur: Author:	Dalia Attia
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Attia, D. (2020). Optimization of heterogeneous employee scheduling problems [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/5291/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/5291/
Directeurs de recherche: Advisors:	François Soumis, & Guy Desaulniers
Programme: Program:	Doctorat en mathématiques

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Optimization of heterogeneous employee scheduling problems

DALIA ATTIA
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Mai 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Optimization of heterogeneous employee scheduling problems

présentée par **Dalia ATTIA**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Nadia LAHRICHI, présidente
François SOUMIS, membre et directeur de recherche
Guy DESAULNIERS, membre et codirecteur de recherche
Antoine LEGRAIN, membre
Quentin LEQUY, membre externe

DEDICATION

To my beloved husband and kids,
to my mother and to my father's soul.

ACKNOWLEDGEMENTS

Pursuing my PhD is a dream coming true. It would not have been possible without the guidance and support of many people. First of all, I would love to thank my supervisors who gave me this opportunity, Prof. Dr. François Soumis and Prof. Dr. Guy Desaulniers. You taught me, helped me and encouraged me a lot through the past years.
I am also grateful to Prof. Dr. Nadia Lahrichi, Prof. Dr. Antoine Legrain and Prof. Dr. Quentin Lequy for accepting to be part of the jury.
Furthermore, a special thanks to Prof. Dr. Mohamed Ismail, whose encouragement was the first spark of this journey.

I would like to thank Kronos Canadian Systems and the Natural Sciences and Engineering Research Council (NSERC) for supporting the research with needed funds.

A very special gratitude goes out to all my family. To my beloved husband, Mina, without you I would not have succeeded, your support cannot be expressed in words, you have always believed in me, thank you. To my kids, Mark, Maria and George, I love you. To my guardian angel, my mother Fadia, you are my inspiration. To my father Waguih, you are always my idol, especially on engineering and mathematics, I miss you. To my cousins, Mary, Samer and Michael, thank you for your help and encouragement. And before all to God, who gave me strength and grace.

RÉSUMÉ

Le problème de planification d'horaires du personnel consiste à créer les horaires de travail des employés d'une organisation. Le nombre d'employés requis par unité de temps, appelé la demande en employés par période, est donné pour un horizon de planification. Différentes règles et contraintes régissent l'élaboration des horaires des employés. Ces règles dépendent des besoins de l'organisation, des contrats des employés et de la convention collective de travail.

Le problème de planification est dit hétérogène quand il concerne des employés ayant des qualifications différentes, habituellement, dans le cadre d'un problème de planification d'employés multi-tâches ou multi-départements. Dans un contexte multi-départements avec transferts entre départements, un quart de travail peut être effectué dans son ensemble dans un département, ou un transfert de département peut avoir lieu au sein du quart de travail lorsque l'employé a les qualifications requises.

Lorsque les transferts sont autorisés, le nombre de quarts de travail possibles par employé devient énorme. L'optimisation d'un tel problème est souvent essentielle pour le succès de l'organisation. Par contre, sa résolution directe comme un programme linéaire en nombres entiers s'avère impossible pour les grandes instances.

Dans la première partie de cette thèse, nous proposons une heuristique de décomposition en plusieurs phases (MP-DH) pour le problème de planification des employés avec transferts. Dans ce problème, la sous-couverture et la sur-couverture sont acceptées mais pénalisés dans la fonction objectif. Un *département d'origine* est introduit pour chaque employé où l'employé doit travailler la majorité de son temps. En plus, il/elle peut être qualifié(e) pour travailler dans plusieurs autres départements. La première phase commence par déduire la demande en employés qui ne peut pas être couverte par les employés du département. Ces pièces de demandes extraites sont appelées *intervalles critiques*, car elles nécessitent des employés transférés d'autres départements pour y travailler. Cela se fait en résolvant un programme en nombres entiers de planification d'horaires de personnel anonyme pour chaque département séparément, puis en extrayant la demande non-couverte qui définit un ensemble d'intervalles critiques.

Pour chacun des intervalles critiques, la deuxième phase choisit un département qui lui attribue la responsabilité de transférer un de ses employés pour travailler pendant cet intervalle critique. Ceci est accompli en résolvant un autre programme en nombres entiers de planification d'horaires de personnel anonyme avec transferts, pour *un seul* jour, pour chacun des jours de l'horizon. La décomposition journalière rend la taille du problème gérable spécialement pour les grandes instances. Cette phase se termine par la migration de toute demande d'un département d_1 couverte par un employé d'un département d_2, formant la *demande de transfert d'employé* de d_2 vers d_1.

Finalement, pour chaque département, la troisième phase résout un programme en nombres entiers de planification d'horaires de personnel mono-départemental avec transfert. La demande utilisée est la nouvelle demande résultant de la migration de toutes les demandes de transfert pendant la deuxième phase.

L'heuristique MP-DH a réussi à décomposer le problème de planification d'employés multi-départements en plusieurs, plus petits, problèmes de planification d'employés mono-départementaux, ce qui a permis de réduire de beaucoup les temps de calcul et de transformer les grandes instances non résolubles en instances résolubles avec une légère baisse dans la qualité des solutions obtenues.

Chacune des trois phases de MP-DH utilise le parallélisme. Dans la première phase, les programmes en nombres entiers des départements s'optimisent en parallèle. La deuxième phase exécute chaque problème journalier en parallèle. Enfin, la troisième phase optimise chaque département en parallèle. À la fin de chaque phase, tous les résultats des problèmes parallèles sont fusionnés pour former la solution finale.

Dans les tests réalisés pour l'heuristique MP-DH, les deux premières phases sont extrêmement rapides, tandis que la troisième phase peut atteindre deux heures de temps de résolution pour les grandes instances. Pour pallier à cet inconvénient, nous présentons une *heuristique hybride* dans la deuxième partie de la thèse, visant à réduire fortement le temps d'exécution de la troisième phase tout en conservant la qualité de la solution.

L'heuristique hybride utilise deux modèles de manière interchangeable afin de résoudre la troisième phase le plus précisément et rapidement possible. Le premier modèle est celui déjà presenté pour la troisième phase du MP-DH que nous appelons le *modèle de base*. Le second est un problème de planification d'horaires du personnel mono-département avec transfert

semi-anonyme que nous appelons le *modèle semi-anonyme*. La version semi-anonyme réduit le nombre d'employés pour lesquels les horaires sont optimisés et remplace les quarts des employés restants par un ensemble de quarts anonymes agrégés, puis résout le problème pour les employés restants par la suite. L'heuristique hybride commence par résoudre le modèle de base. Si après un délai donné, l'écart d'optimalité est supérieur à un seuil donné, la résolution de modèle de base est annulé et une version semi-anonyme est résolue. Cette opération est répétée jusqu'à ce que tous les horaires des employés soient optimisés. L'heuristique hybride a réussi à réduire le temps d'exécution de la troisième phase jusqu'à 87% en moyenne, tout en perdant seulement 4 % dans le coût de la solution en moyenne.

Dans la troisième partie de la thèse, nous abordons une version différente du problème de planification d'horaires de personnel, soit le problème de planification d'horaires de personnel multi-tâches, où ni les transferts ni la sous-couverture ne sont autorisés. À la place de la sous-couverture, des quarts anonymes appelés *open-shifts* sont utilisés pour couvrir la demande incouvrable par aucun employé. Nous développons une métaheuristique parallèle de recherche à grands voisinage (LNS) pour ce problème. Le concept de *sub-scope* est utilisé comme unité de décomposition dans l'algorithme LNS. Un sub-scope est défini comme: un sous-ensemble d'employés, un sous-ensemble de tâches et un sous-ensemble continu de l'horizon du problème. L'heuristique LNS est définie par des procédures de destruction et de réparation. Notre procédure de destruction choisit des sub-scopes, entraînant un coût élevé, à détruire. Lorsqu'un sub-scope d'une solution est détruit, tous les quarts travaillés pendant l'horizon du sub-scope par un employé appartenant au sub-scope, pour l'une des tâches du sub-scope, sont supprimés de la solution.

Les coûts principaux affectant la fonction objectif sont les suivants: le coût de la sur-couverture, le coût d'utilisation des open-shifts et la pénalité pour la violation des heures de travail minimales des employés. La procédure de destruction se concentre donc sur la destruction des sub-scopes entraînant de tels coûts dans une solution donnée. Après la destruction des sub-scopes d'une solution, la procédure de réparation reconstruit une nouvelle solution améliorée. La procédure de réparation que nous proposons résout un programme en nombres entiers de planification d'horaires de personnel multi-tâches pour les sub-scopes déjà détruits. Les procédures de destruction et de réparation sont répétées séquentiellement jusqu'à ce que la condition d'arrêt soit atteinte.

La procédure de destruction parallèle détruit plusieurs sub-scopes disjoints, puis chaque sub-

scope est réparé dans un fil (thread) parallèle différent. Nous comparons l'heuristique présentée avec le modèle exact résolu dans le système commercial WFC par Kronos Inc. Les résultats expérimentaux montrent qu'en moyenne, l'algorithme LNS parallèle peut réduire les temps d'exécution jusqu'à 80% et améliorer les coûts des solutions jusqu'à $1,8\%$.

ABSTRACT

The employee scheduling problem consists of creating working schedules for an organization staff. The number of required employees per time unit, called employee requirement per period, is given for the full problem horizon. Different rules and constraints govern an employee scheduling problem, these rules depends on the organization needs, employees contracts and the collective labor agreement.

A heterogeneous employee scheduling problem deals with employees having different working skills, usually within a multi-job or multi-department employee scheduling context, where one employee can be qualified for several of the organization activities, and can work for any activity he/she is qualified for. One working shift can be accomplished in a single department, or a department transfer can take place within a shift when the employee has the required skills.

When a department transfer within a shift is allowed, the number of possible working shifts per employee becomes huge. Optimizing such heterogeneous employee scheduling problem is often essential for organizational success. However, solving such problems directly as a mixed integer linear program (MILP) is intractable for large instances.

In the first part of this thesis, we propose a multi-phase decomposition heuristic (MP-DH) for the employee scheduling problem with inter-department transfers. In this problem, the concept of *department of origin* is introduced, where each employee is qualified to work in several departments, but he/she has exactly one department of origin, where the employee should work the majority of his/her time. The first phase starts by extracting from each department employee requirement, the uncoverable requirement parts by internal employees, i.e. if only the department internal employees can work. These extracted requirement parts are called *critical intervals*, because they need transferred employees from other departments to fulfill them. This is done by solving an anonymous employee scheduling problem modeled as a MILP for each department apart, before extracting the uncovered requirement parts that form the set of critical intervals.

For each of the critical intervals, in the second phase, one department is chosen to assign it the responsibility of fulfilling this critical interval requirement, i.e. to transfer one of its employees to work during the critical interval. This is accomplished by solving a *one-day*

anonymous employee scheduling problem with inter-department transfers for the critical intervals modeled as a MILP, for each of the problem horizon days. The day decomposition renders the problem size manageable in computer memory, especially for large instances (up to 25 departments). This phase ends by migrating any department d_1 requirement covered by an employee from department d_2, building a new *employee transfer requirement* from d_2 to d_1.

The third phase solves, for each department, a mono-department employee scheduling problem with derived inter-department transfers as a MILP. The input to the third phase is the new final requirement resulting from the requirement migration of phase two.

The MP-DH heuristic succeeds to decompose the multi-department employee scheduling problem into several mono-department employee scheduling problems to save substantial computational time. This allows to solve large instances while not deteriorating much the solution quality.

Each phase of the MP-DH algorithm uses parallelism. In the first phase, all department MILPs and post-processing are accomplished in parallel. The second phase runs all single-day problems in parallel. Finally the third phase optimizes all department problems in parallel. At the end of each phase, all parallel problem solutions are merged to form the final solution.

In the reported computational experiments, we observe that the first two phases are solved extremely fast compared to the third phase. The size of the solved MILPs in the first two phases is not proportional with the size of the optimized instance, while the third phase MILP size is. To overcome the computational issues of the third phase we present the *hybrid heuristic* in the second part of the thesis. The hybrid heuristic aims at greatly reducing the MP-DH third phase computational time, while maintaining the solution quality.

The hybrid heuristic uses two models interchangeably in order to solve the third phase as accurate and as fast as possible. The first is the third phase MILP of the MP-DH algorithm, which we call the *basic model*. The second is a semi-anonymous employee scheduling problem with derived inter-department transfers modeled as a MILP that we call the *semi-anonymous model*. The semi-anonymous version reduces the number of employees for whom the schedules are optimized, and replaces the remaining employee shifts by a set of aggregated anonymous shifts. Once such a model is solved, the schedules of the selected employees are fixed and

the algorithm moves on to solving another MILP where another set of employees must be scheduled. The *hybrid heuristic* starts by solving the basic model, then if, after a given time limit, the MILP optimality gap is higher than a given threshold, the resolution of the basic model is stopped and a semi-anonymous version is solved. This is done repeatedly until all employee schedules are optimized. The *hybrid heuristic* succeeded in reducing on average up to 87% of the third phase computational time while only loosing 4% in the solution quality.

In the third part of the thesis, we tackle a different employee scheduling problem variant: the multi-job employee scheduling problem, where neither transfers nor under-coverage is allowed. Instead, anonymous shifts called *open-shifts* are used to cover any unavoidable under-coverage. The three main costs composing the objective function are: Over-coverage cost, open-shift usage cost, and minimum employees working hours violation penalty. A parallel large neighborhood search (LNS) metaheuristic for the multi-job employee scheduling problem is developed. Where a *sub-scope* denotes: a subset of the employees, a subset of the jobs and a continuous subset of the problem horizon. The LNS heuristic is defined by destroy and repair procedures. Our destroy procedure chooses sub-scopes coupled with a high cost in the objective function to be destroyed. When a solution sub-scope is destroyed, all shifts, occurring within the sub-scope horizon and worked by an employee belonging to the sub-scope, for one of the sub-scope jobs, are removed from the current solution schedule. Once the solution sub-scopes are destroyed, the repair operator tries to build an enhanced solution. Our proposed repair operator solves a MILP restricted to the destroyed sub-scopes.

The parallel LNS destroy operator creates several disjoint sub-scopes, then each sub-scope is repaired in a different parallel thread. We compare the presented heuristic with the formal MILP solved within the commercial system WFC for Kronos Inc. Experimental results show that the parallel LNS algorithm can save up to an average of 80% in the computational time and 1.8% in the solution cost.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

MILP	Mixed integer linear program
ESP	Employee scheduling problem
ESP-IDT	Employee scheduling problem with inter-department transfers
ESP-DIDT	Employee scheduling problem with derived inter-department transfers
MP-DH	Multi-phase decomposition heuristic
SA-ESP-DIDT	Semi-anonymous employee scheduling problem with derived inter-department transfers
SA heuristic	Semi-anonymous heuristic
HH	Hybrid heuristic
MJ-ESP	Multi-job employee scheduling problem
LNS	Large neighborhood search
PLNS	Parallel large neighborhood search
ALNS	Adaptive large neighborhood search
WFC	Workforce Central

CHAPTER 1 INTRODUCTION

Employee scheduling is an important process for several industries, where each employee is assigned his working shifts for a predefined working time-span called *horizon*. Small savings in every employee schedule can result in huge benefits for the organization. Savings can be seen from a cost perspective, where minimizing the number of employee working hours can save lots of money. But nowadays many other optimization perspectives are considered. Employee satisfaction as well as customer satisfaction are of great importance for an organization to grow.

When an employee is satisfied, his performance is higher than when he works under pressure or unsatisfactory conditions. An employee is satisfied if his choices of working days and days off are respected, as well as his needed rest duration between successive shifts, his requested shift duration, and the minimum and maximum weekly working hours. An employee working a side job, just for extra pocket money, has different needs than an employee working to maintain his family, and need to work full time to be able to sustain his family needs. Customer satisfaction is directly proportional with the organization success, unsatisfied customer will probably stop dealing with the unsatisfying organization. One of the parameters affecting customer satisfaction is the waiting time for receiving the needed service. A customer standing too long on a cashier line, or waiting too long on a phone call may simply decide not to buy his goods and skip the line, or cancel his asked service and hang up the call. Customers will wait a lot if the number of available employees is less than the actual needed number of working employees, which we call in our study *under-coverage*.

With the growth of an organization (companies, retailers, hospitals, ...), the number of employees increases, along with the size and cost of scheduling their shifts. As a result, any small percentage of cost reduction, within each employee shift schedule, is reflected in a huge benefit for the organization. Over the years, employee scheduling problems become more complex. Beside larger instances due to the growth of the companies, additional constraints need to be handled to guarantee employee and customer satisfaction. Also the variation of the work load along the hours of the day (or week or month) makes the shift placement a complicated task, where the morning-evening-night shift pattern is not suitable anymore.

1.1 Problem and motivation

In our study, we solve two different variants of the employee scheduling problem. The first one is called the multi-job employee scheduling problem, where every employee has several qualifications, thus can work on several jobs. For each of the employee working days, the employee is assigned at most one shift for one of his qualified jobs, without exceeding the employee maximum working hours. The second variant deals with organizations having multiple departments, where employees can work for two different departments within the same shift, yielding what we call inter-department transfers. Allowing transfers between departments within a single shift gives a preference for *multi-department* in the problem name over *multi-job*. These employee scheduling problems are usually found in service centers and retail stores.

1.1.1 Multi-job employee scheduling problem

For the multi-job scheduling problem, employees have multiple qualifications along with high flexibility with respect to their shift preferences. Employees can specify their own preferred maximum and minimum weekly working hours, their maximum and minimum single shift duration, their days off, minimum rest duration between shifts, as well as their shift starting time. Some employees prefer to start their shifts early, others late. A single shift serves exactly one job, an employee cannot work for a job that he is not qualified for, and can work at most one shift per day.

Under-coverage is not allowed, instead, unassigned shifts are created in order to cover any unavoidable under-coverage when no employee is found to accomplish the shift. The unassigned shifts, called *open shifts*, have their own minimum and maximum duration rule. Open shifts are supposed to be assigned to employees who only work on demand, thus they cannot be of arbitrary size, and they can even be a source of some over-coverage cost.

Over-coverage is a result of having extra working employees when they are not really needed. Over-coverage is caused by either personalized shifts or open shifts. An open shift covering, e.g. one hour of unavoidable under-coverage and causing an additional three hours of over-coverage, in case of minimum shift duration of four hours, may be preferred in our problem over the one hour under-coverage, as over-coverage means some extra spent salary, while under-coverage results in customer dissatisfaction.

Several costs are taken into account in the multi-job employee scheduling problem, shaping together the quality of a schedule. First there are personalized shift costs: employee working hours are mapped to a convex cost function. Opposite to the linear cost function, convex cost function guarantees fair workload balance between employees. An open shift is more expensive than its respective personalized shift, in order to prioritize assigning shifts to the current contracted employees. Second, a cost is incurred for violating the preferred minimum number of working hours of an employee. Employees maximum working hours limit is maintained with hard constraints, but violating employee preferred minimum working hours is penalized in the objective function using a convex function of the deviated number of hours. Again the convex function helps in equally distributing any dissatisfaction among employees. Third, there are over-coverage cost. Under-coverage is not allowed in our model, but over-coverage is allowed and penalized in order to minimize its presence. Also any existing over-coverage per period is mapped to a convex function, so that peaks of over-coverage at a single time period is avoided, and the over-coverage, if any, is spread among the entire working horizon as much as possible.

1.1.2 Employee scheduling problem with inter-department transfers

In this problem variant, we are concerned with a feature imposed by institutional growth, namely: inter-department transfers. This feature is supported by the quote "*A key opportunity inherent in solid scheduling, is in using employees in different functions, as needed.*" from Thompson [72].

Along with fluctuating demand and different employee qualifications and preferences, managers need to fulfill all departments' requirements with the least cost.

Permitting an employee to work in many departments would minimize the total number of required employees, and eventually reduces the cost. Meanwhile, changing the employee's department transition time and cost must be considered. Thus the scheduling process is not simplified, but complicated.

In this problem variant, under-coverage is permitted but penalized in the objective function as well as over-coverage. A new parameter for each employee is introduced, namely the *home department*, which is the main department to which the employee is attached. The employee qualified departments are all departments where the employee can work, including

the home department. Only transfers to or from the home department are allowed, and a maximum of one transfer per shift is permitted. Another allowed shift type is the *external shift*, where the whole shift starts and ends in a department other then the home department.

Several attempts have been made in the literature for optimizing different employee scheduling problems with different sets of constraints. One conclusion can be found in all studies: formal methods cannot solve large problem instances in reasonable time. In order to solve such instances, heuristics, constraints relaxation, and/or problem decomposition must be used, and a tolerance on the solution quality must be accepted.

1.2 Research objectives

In this thesis, our first objective is to develop a decomposition-based heuristic for the employee scheduling problem with inter-department transfers. The presented decomposition scheme transforms the multi-department problem into several mono-department problems. Decomposition introduces some loss of information leading to a degradation in the solution quality. In this problem, the needed transfer information is lost with the department decomposition. To overcome this drawback, a pre-processing procedure creating a set of expected needed transfers between each department is presented. Each department expected transfers are then used in its mono-department employee scheduling problem with derived transfers optimization.

The presented decomposition-based heuristic transforms the multi-department employee scheduling problem into several mono-department employee scheduling problems with expected transfers, which means that one large problem is decomposed into m smaller sub-problems, where m is the number of departments. But each mono-department problem remains a large, highly constrained problem. Thus our second objective is to establish a hybrid heuristic with the main objective of minimizing the computational time of the employee scheduling problem with expected transfers.

The third objective is to create a parallel large neighborhood search heuristic for the multi-job employee scheduling problem. Meta-heuristics are among the highly scalable solution methods, as they can solve very small sub-problems in each iteration to find a new solution in the current neighborhood. This makes meta-heuristics suitable for optimizing large problem

instances. Additionally, parallelism helps investigate the solution space faster by using the computer multi-processing capability. Large neighborhood search is best suited for problems that can be easily decomposed, in order to create smaller sub-problems to be destroyed then repaired. This characteristic can be found in the multi-job employee scheduling problem, where a sub-problem is defined by a sub-horizon, a subset of jobs and a subset of employees.

1.3 Thesis outline

This thesis is organized as follows. Chapter 2 reviews the literature for the employee scheduling problem. Chapter 3 discusses the organizing of the following main chapters. Chapter 4 presents the decomposition based heuristic for the employee scheduling problem with inter-department transfers. In Chapter 5 we develop a hybrid heuristic accelerating the mono-department employee scheduling problem with transfers. Chapter 6 develops the parallel large neighborhood search heuristic for the multi-job employee scheduling problem. The overall work is further discussed in Chapter 7. And we conclude in Chapter 8.

CHAPTER 2 LITERATURE REVIEW

In the literature, the employee scheduling problem is studied in diverse application areas. Each area imposes its unique set of hypotheses and constraints. This makes it hard to find many previous works with exactly similar specifications as the problems of interest in our work. We first present a general review of the employee scheduling problem in various areas, then elaborate on works sharing some specifications with ours.

2.1 Employee scheduling problem specifications

We start by presenting the different problem specifications found in the literature for the employee scheduling problem. Afterwards, we discuss the different solution methods, including exact and heuristic methods.

2.1.1 Shift, days-off and tour scheduling

When it comes to the creation of employee working timetables, one of the next problems is considered (Baker [5])

- Shift Scheduling: The simplest form of scheduling. For a single day horizon, we need to create employee shift covering the demand for each time period along the day.
- Days-Off scheduling: In a weekly problem usually employees work less than 7 days per week. Employee days-off are scheduled to control the employee availability along the week.
- Tour scheduling: The mix between shift scheduling and days-off scheduling. Tour scheduling is the most frequent in the literature (Van den Bergh et al. [73]).

2.1.2 Cost

Different costs, to be minimized, are associated with different problems in the literature. For example:

- Minimizing the employee salary. Employee salary can increase with skills, seniority, or use of overtime.

- Adding penalties for any under or over-coverage with respect to the workforce requirement, (Munezero [57]).
- Transportation costs or transition costs are found when employees need to move in order to continue their shifts in a new site (Bard and Wan [8]).
- Penalties for employee dissatisfaction when employee preferences are not respected (Bürgy et al. [12]).
- Minimizing re-scheduling changes. In some areas, periodic re-scheduling is mandatory in order to accommodate any employee sudden unavailability or new workforce demand. When re-scheduling, schedule perturbations should be minimized, (Bard and Purnomo [7]).

2.1.3 Problem constraints

Several sets of constraints are found in the literature for the employee scheduling problem. Here we present the main and usually used constraints.

Employee specifications

Many characteristics can be associated with each employee in an organization.

- The contract type: Full time employees or part time employees. An optimization problem can involve one or more contract types, controlling the number of maximum and minimum employee working hours (Bard and Wan [8]).
- Specific skills: The assignment of an employee to a task which he is not qualified can be either forbidden or allowed with a penalty. Heterogeneous multi-job employee scheduling problem always involves checking the employee skills (Bürgy et al. [12]).
- Employee seniority: Seniority, whether in term of age or experience, can give privileges to an employee, e.g. his/her preferences are prioritized (Volgenant [74]).

Shift organization

- Timing: Maximum and minimum shift duration. Most of the times it is imposed by enumerating only the eligible shifts.
- Stint based shifts: For some application areas, employee shift must follow special stint, e.g. Day-Day-Evening-Evening-Night-Night shift sequence (Ernst et al. [33]).

- Cyclic vs acyclic: Some problems deal with cyclic demand, where the same demand is repeated every week (or any horizon duration). Other problems are acyclic, where the demand is changed continuously (Millar and Kiragu [55]).

Days-off

Some organizations operate seven days a week, while the rules restrict the number of weekly worked days and hours per employee, introducing the days-off scheduling constraints (Baker and Magazine [6]). Examples of days-off constraints include:

- Weekly consecutive days-off.
- One day, of each employee days-off, must be on a week end.
- At least one complete week end for each employee every month.
- Predetermined employee days-off (Munezero [57]).

2.1.4 Demand modeling

Non-overlapping shift employee scheduling problems are easy to solve as per Baker [5]. But problems involving demand fluctuation must use overlapping shifts.

- Stable demand: Where three 8-hour shift pattern is used (day-shift, evening-shift and night-shift). The demand is constant during each shift (Bard and Purnomo [7]).
- Fluctuating demand: This model is found in call centers and service providers, where the number of incoming service requests varies all over the day (Dahmen and Rekik [20]).

Demand coverage constraints

Demand coverage is achieved by assigning enough employees in each time period for the planning horizon. In the literature, this is accomplished by:

- Hard constraints: Explicitly add a constraint to guarantee the demand coverage. This is applied when under-coverage is not allowed (Bard and Wan [8]).
- Soft constraints : Adding penalties to the objective function minimizing any under-coverage, (Munezero [57]).

Uncertainty

In employee scheduling, two types of uncertainty can be found:

- Demand uncertainty: In many cases, the work load distribution is an estimation based on previous demand data and marginal analysis (Easton and Rossin [31]). Deviation in the incoming demand can happen, especially in the service organizations like call centers and retail stores. Easton and Rossin [31] use a probability distribution to model the required employee demand for each time period of the horizon.
- Employees arrival uncertainty: Late arrival or sudden absence of employees can cause a degradation in the service level. Increasing the workforce requirement based on the expected late/absence would help in enhancing the service.

Michon-Lacaze [54] deals with both types of uncertainty, calling them perturbations, either demand perturbations (sudden increase in the demand) or employees arrival time perturbation (late employee). Shifts are created in a way which facilitates future possible changes to cover a sudden perturbation. Given the perturbation probabilities, Michon-Lacaze [54] updates the problem demand then solves the updated problem using an integer linear program.

2.1.5 Application areas

Employee scheduling is a popular problem found in diverse application areas. We cannot restrict the optimization of employee scheduling problem only in the next contexts, but we list the most popular ones in the literature. The reader is encouraged to check Defraeye and Van Nieuwenhuyse [25] for scheduling papers dealing in various application areas.
Popular areas where employee scheduling is applied include health care, transportation, services, and manufacturing. Optimizing nurse rosters in hospitals is of great importance in the health industry. In the transportation industry, airlines, bus, railways and subways personnel schedules are optimized. Service organizations and retailers also use employee scheduling heavily in order to optimize the number of service providers with respect to the number of customers. Examples of the service organization are call centers, emergency services like ambulance and fire services, shipping and mailing service providers, and chain retailers (Canon [14], Bard and Wan [8], Talarico and Duque [71]). In the manufacturing area, employee scheduling is used in order to optimize the machines' utilization and maximize the production with the minimal number of employees.

2.2 Mathematical optimization

Several methods have been proposed to solve an employee scheduling problem. The method is chosen according to the problem characteristics, like the expected problem size and acceptable computational time.

2.2.1 Set Covering Model

Mathematical programming is one of the oldest methods used for solving the employee scheduling problem. As previously stated, recent problems are different from the early employee scheduling problem. One of the first articles dealing with employee scheduling was presented by Edie [32]. The author was trying to reduce the costs associated with the collection of vehicle tolls at *The Port of New York Authority tunnels and bridges*. A sophisticated statistical model was developed to estimate the necessary number of working toll booths for every half hour of the day. Edie's last step was to create employee schedules: assign toll collectors to shifts, so that the toll booth-hour requirements, and the collectors' relief time are maintained. "*This is largely a trial-and-error problem, and preparation of such schedules may be very time-consuming when the objective is to make the schedule as efficient as possible.*" Edie [32]. This is how Edie tackled the employee scheduling problem at that time. This model shows more than 98% satisfaction, as per the author's measurement "*The ratio of the number of collectors required by the booth-hour to the number supplied by the schedule*" Edie [32].

In Dantzig [23], the author modeled Edie's employee scheduling problem as a set covering problem. The model minimizes the cost associated with the whole set of employees, under constraints ensuring the existence of enough employees satisfying total demand, and suitable relief time for each person. This classical set covering optimization model still plays an essential role in many recent problem solution algorithms.

2.2.2 Implicit programming

More than forty years later, Bechtold and Jacobs [10] presented a new integer programming model with fewer variables then Dantzig [23]. Bechtold and Jacobs [10] qualified their model as implicit. They implicitly represent the break assignments in all shifts by associating the break variables with the planning periods instead of the shifts. This model shows significant reduction in the number of variables in comparison with the old set-covering model. However, the validity of the proposed model relies on a set of nine assumptions that can be found in

many real-life problems.

Aykin [3] relaxed most of these assumptions and proposed a generalized implicit integer programming model allowing multiple breaks for employees. In the comparative study Aykin [4], Aykin's implicit model was found to be 66% faster than Bechtold and Jacobs' model, on average.

Dahmen et al. [21] generalize the shift scheduling implicit model to the multi-activity context, with adapted forward and backward constraints.

2.3 Metaheuristics

Metaheuristics form an important family for solving optimization problems. They have many advantages like they can fit for nearly any problem, they are simple to implement, and can produce a feasible solution, often of high quality, for problems where exact methods can spend a lot of time without returning any solution. Their main disadvantage is that they cannot guarantee optimality.

Metaheuristics are categorized under two famous classes: population-based methods (evolutionary algorithms) and trajectory-based methods (neighborhood search). Population-based methods generally consist of an interaction between many solutions (or semi-solutions) to introduce another set of new solutions (new generation). The basic idea is to merge the best part of every parent-solution in order to produce a better child-solution. Example of such algorithms are genetic algorithms and scatter search. Aickelin et al. [1] introduce a genetic algorithms evolutionary technique by adding a selection step and an evolution step to the search technique for nurse scheduling. Burke et al. [13] present a scatter search algorithm for nurse rostering. Trajectory-based methods start with a feasible solution, and move forward to another solution by making a simple change to the current solution. We are more concerned about the parallel neighborhood search, specifically parallel large neighborhood search. Thus we review both the large neighborhood search and the parallel metaheuristics with more details in Sections 2.3.2 and 2.3.3, respectively.

2.3.1 Neighborhood search

For a combinatorial optimization problem P, neighborhood search is an iterative process, defining a set of neighbor solutions $N(s_c)$ for a current solution s_c. Let S be the feasible

solution space. Then $N(s_c) \subset S$. A cost function $f(s)$ maps each solution s to its cost, $f : S \to \mathbb{R}$. When minimizing the cost, we search for s^* where $f(s^*) \leq f(s) \quad \forall s \in S$.
In neighborhood search, starting with an initial solution s_0, at each iteration i, the neighborhood $N(s_i)$ is computed for the current solution s_i, and the cost for each solution $s \in N(s_i)$ is calculated. We denote by the neighborhood design the algorithm of calculating the neighborhood $N(s_i)$, and by neighborhood exploration, the process of calculating the cost function of each solution in the neighborhood. Finally the neighbor solution with the least cost is chosen to be the new current solution: $s_{i+1} = argmin_{s \in N(s_i)}\{f(s)\}$. If $f(s_{i+1}) > f(s_i)$, then no improvement is possible and the search ends, returning the last best solution s_i.

Such neighborhood search leads to local optima, not guaranteed to be a global optima. Several heuristics are designed to escape local optima, e.g. by accepting new solutions having a cost greater than or equal to the current solution cost $f(s_{i+1}) \geq f(s_i)$. In this case, other constraints are added to guarantee the search convergence. In the tabu search metaheuristic by Glover [36], changes made to the solutions are preserved in a tabu list for a given number of iterations. Changes existing in the tabu list are forbidden to be re-applied in order to forbid cycling and redirect the search toward new area in the solution space. For the simulated annealing heuristic of Kirkpatrick et al. [44], the probability of accepting a deteriorating solution decreases over time.

2.3.2 Large neighborhood search

A Large Neighborhood Search (LNS) metaheuristic was first introduced by Shaw [69] for solving the vehicle routing problem. The core idea of LNS is to replace the *neighborhood design* and the *neighbors solutions exploration* by *destroy* and *repair* operators, respectively. The destroy operator chooses a subset of the current solution to be destroyed, then the neighborhood exploration is replaced by a repair operator, which is an algorithm to re-construct the destroyed parts of the solution, in a manner to ensure as much as possible an improvement in the current solution cost. For a mathematical program, the destroy operator can be seen as a relaxation of a subset of the problem variables, and the repair operator is re-assigning values to the relaxed variables intelligently.

Shaw [69] presents LNS as a heuristic combining the power of local search and constraint programming, where the destroy and repair operators produce together a new solution from an old solution as in local search moves. At the same time the proposed repair operator in Shaw [69] is performed by constraint programming, which is a short-cut to achieve the

optimal neighbor solution without exploring the whole neighborhood.

In LNS, the deterioration of a portion of the current solution and its repair become time consuming due to its large size. Thus the number of iterations per second is small compared to other metaheuristics. However the iterations become more powerful and achieve higher improvement (Shaw [69]). In Ropke and Pisinger [63], the destroy operator randomly chooses a destruction degree up to half of the problem variables. The authors confirm that very good performance is observed even if the needed computation time per iteration is much higher compared with standard metaheuristics. Shaw [69] proposed to gradually increase the degree of destruction. Other works limit the maximum number of destroyed variables to a predefined value, acting as the maximum hamming distance between the current solution and the new neighbor solution (Della Croce and Salassa [26]). The power of each move is the core idea behind LNS. However, precaution is to be taken not to destroy an extremely large part of the problem at each move and turning the metaheuristic into successive re-optimization, leading to poor-quality solutions (Pisinger and Ropke [61]).

It is interesting to cite the ruin and recreate (R & R) method presented in Schrimpf et al. [68] published after Shaw [69]. R & R is similar to the LNS idea, where iteratively parts of the solution are ruined then re-constructed. Schrimpf et al. [68] presents the ruin and recreate strategy for the vehicle routing problem and the telecommunication network optimization problem. For the vehicle routing problem, radial, random and sequential ruining procedures are used. In radial ruin, a random node is chosen along with its nearest n nodes, and all of them are removed from the current route. Random removal is a simple random choice of n nodes to be removed from the solution route. And finally the sequential ruin removes n sequential nodes from a randomly selected round trip route. The recreate step uses a greedy best insertion for simplicity. For the network optimization problem, simple ruining technique is used by removing some demand and downsizing the links bandwidth. Then the re-create operator uses a collective best insert procedure. Single best insert is done by inserting the cheapest path for the chosen demand, while collective best insert tries to add several best paths simultaneously. In the paper, simulated annealing (Kirkpatrick et al. [44]), threshold acceptance (Dueck and Scheuer [30]) and greedy acceptance are all tested for the ruined and re-created solutions acceptance strategy.

The destroy and repair operators are the main factors affecting the LNS behavior. They are responsible for each iteration new solution quality. Destroying parts of the solution that

should not be changed will lead to poor or no improvement. Weak repair operators can miss better solutions. Next we introduce the destroy and repair operators used in the literature within different problem contexts.

Destroy heuristics

Three types of destroy strategies are found repeatedly in the literature: the random destroy, worst destroy and related destroy strategies. Next we present them along with other strategies from the literature.

(i) **Random destroy**

Random choice of variables (or problem elements) to be destroyed is the simplest destroy operator. Also considered as a diversification of the search and used as one of several destroy operators in the adaptive LNS (ALNS). The ALNS uses several destroy/repair operators interchangeably in an adaptive way, as the solution process progresses, the operators that have yielded the best solution improvements have higher chances to be selected at each iteration. Random destroy operators have been used in various LNS algorithms, including those proposed by Pisinger and Ropke [61] and Schrimpf et al. [68] for the vehicle routing problem, Sacramento et al. [66] for the vehicle routing problem with drones, Pisinger and Ropke [60], Ropke and Pisinger [64] and Ropke and Pisinger [63] for the pickup and delivery problem with time windows, Wen et al. [75] for the electric vehicle scheduling problem, Laborie and Godard [47] for the activity scheduling problem, Muller [56] for the resource constrained project scheduling problem, and Godard et al. [37] for the cumulative scheduling problem.

The randomization process can be incorporated within other systematic destroy processes, where a part of the removal process follows a well-defined algorithm and the other part is randomized. Pisinger and Ropke [60] and Ropke and Pisinger [64] use a random coefficient controlling the randomness degree incorporated within the systematic destroy procedure. In Cordeau et al. [15], a *last-random destroy* is used for the technician and task scheduling problem, where a task causing high cost is removed, then another set of random tasks are removed. Also the *whole-team destroy* selects a day randomly, then two technician teams working this day are randomly removed.

(ii) **Worst/Critical destroy**

This destroy operator is considered as an intensification of the search (Pisinger and Ropke [61] and Pisinger and Ropke [60]). Destroying the expensive variables within a

solution and keeping the low-cost variables gives a chance to get a lower overall cost while repairing the solution. Ropke and Pisinger [63] and Pisinger and Ropke [60] use a worst removal for the pickup and delivery problem with time windows. A cost function is defined for each request (pickup/delivery) node and the top q most expensive requests are removed from the solution. Muller [56] uses a *critical-path* removal as one of the ALNS removals for the resource constrained project scheduling problem. Activity paths consuming a lot of time are removed. Dahmen and Rekik [20] use a worst destroy operator for the multi-activity employee scheduling problem.

(iii) **Related destroy**

Also known as Shaw-destroy, as related destroy was first introduced by Shaw [69]. A relatedness measure is defined depending on the nature of the problem and how variables are considered related. The idea behind the related destroy is that it is easy to do an exchange between related variables. For the routing problems, the relatedness is the distance between nodes (Ropke and Pisinger [64] and Pisinger and Ropke [61]). For the pickup and delivery problem in Shaw [69], a related coefficient between nodes considers whether or not two nodes are served by the same vehicle within the current route. For an electric vehicle scheduling problem, Wen et al. [75] devised a neighboring vehicle-schedule based removal operator that relies on the average distance between two scheduled trips. For the technician and task scheduling problem in Cordeau et al. [15], the relatedness between tasks depends on the common skill requirements between them. For the high school timetabling problem, Demirović and Musliu [28] consider the relatedness between the problem resources: rooms, teachers and students, in the destroy operator.

Several variations based on the related removal principle are found in the literature:

- Space removal:
  If the problem can be modeled on a plan, like the traveling salesman problem or the vehicle routing problem (Schrimpf et al. [68]), space removal looks for all the variables inside a disk (or any shape borders) within the plan and destroy them.
- Cluster destroy:
  The cluster destroy operator is similar to the related destroy. Related destroy defines a relatedness coefficient, then a set of related elements are destroyed. In cluster removal, a set of variables are grouped together with a clustering algorithm then destroyed. In the vehicle routing problem (Ropke and Pisinger [64], Pisinger and Ropke [60] and Kovacs et al. [45]) a route is chosen and its nodes are divided into two clusters, one of them is destroyed randomly.

- Time oriented destroy:
  Variables with approximate time dimension are supposed to be related. For the pickup and delivery problem with time windows, requests with close pickup time are considered related (Pisinger and Ropke [60]). Laborie and Godard [47] use the so called *timeWindowNHood* for the activity scheduling problem, all activities starting within a given time interval are removed from the current schedule. Muller [56] introduces a removal strategy for the resource constrained project scheduling problem called *non-peak*. A non-peak time unit has few running activities, activities running during non-peak time periods are removed. For the electric vehicle scheduling problem in Wen et al. [75], trips are considered related if they share the same start or end time.

(iv) **Historical destroy**
Historical removal is based on keeping track of variable values observed in good-quality solutions, then trying to remove variables that are not frequently found in these solutions. Pisinger and Ropke [60] and Ropke and Pisinger [64] present two historical destroy operators for the vehicle routing problem. The first assigns a weight, initially equal to infinity, for each couple of nodes. When a new best solution is found, the weight for all directly connected pair of nodes is updated to this best solution cost. The historical destroy operator removes connected nodes with high weight. The second historical destroy operator assigns a weight for each couple of requests in the problem, a request is a pair of pick-up and delivery nodes, this weight is increased each time these two requests are served by the same vehicle within a new best solution. These weights are used as a historical-relatedness coefficient while applying the destroy operator.

(v) **Most-mobile destroy**
For the resource constrained project scheduling problem presented in Muller [56], the most-mobile destroy operator selects activities with high freedom to relocate. In the paper context, a relocation means an activity can start before its current starting time, or end after its current ending time. When the most-mobile activities are destroyed, the repair operator has multiple ways for rescheduling the activities.

Repair heuristics

The repair operator is a construction heuristic for the partially destroyed solution. While repair operators highly depend on the problem context, some repeated guidelines are found in the literature. Next we highlight some of the most common repair operators.

(i) **Greedy repair**

For every problem type there exists a greedy algorithm to construct a solution. Greedy algorithms are common because they are simple and easy to implement. Depending on the problem type, a more intelligent repair operator would be necessary. Pisinger and Ropke [60], Ropke and Pisinger [63] and Ropke and Pisinger [64] use a simple greedy repair algorithm for the pickup and delivery routing problem, where iteratively a pickup and delivery request is inserted in its cheapest route. Schrimpf et al. [68] use a greedy best insertion for the vehicle routing problem. Sacramento et al. [66] present four greedy problem-oriented repair heuristics for the vehicle routing problem with drones, where the routing problem involves trucks and aircrafts. For the technician and task scheduling problem in Cordeau et al. [15], a score for each unassigned task is formulated out of several parameters. The repair operator assigns a technician team for the task with the highest score.

(ii) **Regret repair**

In a greedy algorithm, to assign a value for a variable we first try all possible values then choose the variable value which gives the best gain (or the least increase) in the objective function. Once this variable is assigned a value it can not be changed. Such method leads to less possible values for the last variables. In the regret repair, the *difference* between the objective function costs when assigning a variable its best and second-best values is calculated and called the *regret value*. The variable with the highest regret is assigned its best value first. Regret repair is parameterized by the level l of the regret. The regret value represents the loss in the objective function if a variable is assigned its l^{th}-best value instead of its best value. The previously stated algorithm is a regret-2, and a greedy algorithm can be seen as regret-1.

Ropke and Pisinger [63], Ropke and Pisinger [64] and Pisinger and Ropke [60] use the regret repair operator with several levels for the pickup and delivery problem. Also Wen et al. [75] use the regret-2, regret-3 and regret-4 repair operators for the electric vehicle scheduling problem. For the resource-constrained project scheduling problem in Cordeau et al. [15], a *wasted skills* coefficient is calculated before assigning a team to perform a task.

(iii) **Operational research tools**

Mathematical optimization can be used to re-optimize the destroyed sub-problem achieving the optimal neighbor solution. Constraint programming is widely used as a repair operator for its ability to find good solutions and is quite fast for small sub-problems. The repair operator presented by Shaw [69] for the vehicle routing problem uses con-

straint programming. For the activity scheduling problem, Laborie and Godard [47] and Godard et al. [37] use constraint programming to repair destroyed schedules. Della Croce and Salassa [26] present variable neighborhood search with large neighborhoods evaluated by an integer program for the nurse rostering problem. Dahmen and Rekik [20] use the tabu search for the employee scheduling problem, with large neighborhoods where a subset of the schedules is destroyed, then reconstructed using integer programming. For the high-school timetabling problem, Demirović and Musliu [28] repair the destroyed solution, using an exhaustive search based on a partial weighted satisfiability problem (maxSAT) formulation.

2.3.3 Parallel metaheuristic

Parallel metaheuristic exploits the computer multi-core and multi-processor capabilities by dividing a process into several disjoint sub-processes and distribute them over the several available processors. Parallelism is famous for its faster execution property. If the problem can be divided into several sub-problems, these sub-problems can be solved in parallel, and their execution is finished in a fraction of the time needed for the sequential execution. Another important property for parallelism is robustness. Especially for approximate solution methods like metaheuristics, parallelism proves to get higher quality solutions over their sequential versions (Crainic [17]).

Two common classifications for parallel metaheuristics are found in the literature. The first one proposed by Alba et al. [2] divides the trajectory-based parallel metaheuristic into three models: *parallel moves model*, *parallel multistart model* and *move acceleration model*:

(i) **Parallel moves model**
The parallel moves model is the simplest way to use parallelism. For each metaheuristic iteration, the different neighbor solutions are evaluated in parallel.

(ii) **Parallel multistart model**
Several metaheuristic runs are executed in parallel. Each run is parameterized by the initial solution, the used neighborhoods and even the used metaheuristic algorithm. After the parallel executions are finished, the best solution out of the different runs is chosen as the parallel metaheuristic final solution.

(iii) **Move acceleration model**
First presented by Kravitz and Rutenbar [46], the move acceleration model is used when the evaluation of the objective function is time consuming and can be decomposed into independent parts. Thus, all parts are evaluated in parallel to accelerate the evaluation

of the move. The *move acceleration model* does not introduce the parallelism into the metaheuristic search technique or the neighborhood exploration, parallelism is only used for the solution cost evaluation.

The second classification is from Crainic and Toulouse [19] and Crainic [17], where the parallel metaheuristics are classified into four categories: *Low-level parallelization*, *Data decomposition*, *Independent multi-search*, and *Cooperative multi-search*:

(i) **Low-level parallelization**
Low-level parallelization or the *functional-parallelism* strategy as per Crainic [17], includes the parallel heuristics which do not change neither the sequential algorithm logic, nor the problem domain. Only independent low-level processes of the metaheuristic are being parallelized in order to reduce the overall computational time.

(ii) **Data decomposition**
Data, domain, or search-space decomposition arises when the problem domain is divided into sub-domains, and each sub-domain is optimized separately on parallel threads. The final solution is obtained by merging the several thread solutions.

(iii) **Independent multi-search**
Several versions of the metaheuristics are run on several independent threads. After all thread executions are finished, the best solution among the threads is chosen. This methodology benefits from the computer parallel capabilities for the search space exploration without adding any intelligence to the search.

(iv) **Cooperative multi-search**
Cooperation empowers the parallel metaheuristic strategies. Whenever a searching thread reaches a significantly enhanced solution, it notifies the other threads about it, the notified threads can benefit from this enhanced solutions in several ways. This step intensify and diversify the search simultaneously. The intensification process happens when all parallel threads search around the same enhanced solution, and the diversification is obtained because each thread has its own neighborhood definition so the search is diversified in several directions.

Beside the mentioned parallel metaheuristic rigid classifications, Crainic and Hail [18] present a wider taxonomy to classify the parallel metaheuristics. The presented taxonomy is composed of three dimensions: *search control cardinality*, *search control and communication*, and *search differentiation*. To classify a parallel metaheuristic one category value for each dimension is selected. This classification is general enough to cover both the trajectory-based

metaheuristics as well as the population-based metaheuristics. The taxonomy categories are as follows:

(i) **Search control cardinality**
Parallel metaheuristic search control cardinality means the number of controllers in the search. If each thread has its own controller, which decides when to stop, when to change any parameter value, when to diversify or intensify and when to cooperate with other threads, then it is a multi-controller parallel metaheuristic or *p-control* "*pC*". If all threads of the metaheuristic are governed by one controller, then it is a *1-control* or "*1C*" parallel metaheuristic.

(ii) **Search control and communication**
The communication between threads is either *synchronous* or *asynchronous*. In the synchronous communication all threads must stop in order to communicate. In the asynchronous communication each thread decides when it can receive or share information. There exist four categories in the control and communication dimension: rigid synchronous " *RS*", knowledge synchronous "*KS*", asynchronous collegial "*C*", and asynchronous knowledge collegial "*KC*". The expression collegial is inspired from the word *colleague*, where all threads are considered like colleagues working together (Crainic [17]).

(iii) **Search differentiation**
The search differentiation dimension indicates whether the thread metaheuristic uses different or similar initial solution (starting point) and search strategy. *SP*: same starting point. *DP*: different starting point. *SS*: same search strategy. *DS*: different search strategy. Summing up, we have four categories in the search differentiation dimension: *SPSS*, *SPDS*, *DPSS* and *DPDS*.

Next we point to some parallel metaheuristic works in the literature for optimization problems. Saviniec et al. [67] present a parallel local search for solving the school timetabling scheduling problem. The authors develop a parallel multistart trajectory-based algorithm. Four metaheuristics are tested: iterated local search, tabu search, simulated annealing and late acceptance strategy. Jin et al. [40] solve the capacitated vehicle routing problem using parallel multi-neighborhood tabu search. Perron et al. [59] present a parallel large neighborhood search for the network design problem. The large neighborhood search destroy operator freezes random parts of the problem, the remaining parts are then optimized using constraint programming as the repair operator. Munguía et al. [58] present a parallel large neighborhood search for mixed integer programs. Several threads optimize overlapping sub-problems.

Thread final solutions are merged, if the merged solution is infeasible, an extra step ensuring the solution feasibility is accomplished. Fiechter [35] solves the travel salesman problem using parallel tabu metaheuristic. The current solution tour is divided into a number of sub-tours, called slices, each sub-tour is optimized in a parallel thread. After a number of iterations, slices are shifted (boundaries are changed) then reoptimized, to give a chance to discover solutions prohibited during the previous slice optimizations. Lahrichi et al. [48] present a new parallel search framework for multi-attribute combinatorial optimization problems called the *Integrative Cooperative Search*.

2.4 Employee scheduling with inter-department transfers

In this section, we discuss some works dealing with similar employee scheduling problem characteristics as the ones we cover in our study. The multi-job scheduling problem and the employee scheduling problem with inter-department transfers are not covered widely in the literature, but many variation of the problem can be found, like the nurse allocation to different units in a hospital during a shift, and the multi-task or the multi-activity assignment problem, where each employee is qualified for a set of tasks/activities, and the daily shift of an employee usually involves different tasks and activities.

Bard and Purnomo [7] provide an integer program for solving a daily nursing shift assignment problem in a hospital, taking into consideration the possibility of the nurse transfers between different units. If no nurses are available for transfer, on-call nurses shall fulfill the extra demand. The integer program uses only five shift types per day: three 8-hour shifts (Day, Night and evening) and two 12-hour shifts (from 7 a.m. to 7 p.m., and from 7 p.m. to 7 a.m.). This model, with 5 shifts per day, is relatively small compared to problems in other application areas, thus the resolution of the mathematical model is not time consuming.

A heuristic merging constraint programming and integer programming for solving the shift creation and task assignment problem is presented in Demassey et al. [27]. Constraint programming is used for the shifts generation step, then an integer program optimizes the task assignment problem.

Bard and Wan [8] tackle the task assignment problem. Employees have to be assigned to tasks for the week days, requirements must be fulfilled, and employee transitions between different workstation groups must be minimized. The authors start by modeling their problem

as a multi-commodity network flow problem, then present three different solution approaches. The first is a greedy algorithm for the task assignment problem. In the second approach, the problem is decomposed into seven daily problems. A tabu search is presented in the third approach. The main reason for providing three different approaches is to finally combine them into one heuristic. Results show that the daily decomposition heuristic is able to provide nearly optimal solutions, while the tabu search needs a good starting solution in order to provide competitive results. Running the tabu search with the decomposition heuristic result as the starting solution gives the best final results.

A couple of years later, Bard and Wan [9] investigate the same problem with one extra restriction: some transitions between departments are prohibited. Two solution procedures are presented: a sequential procedure and an iterative procedure. In the sequential procedure, the workstation groups are clustered, within each cluster (set of workstation groups), the employee transitions are allowed. First, a simplified tour-scheduling problem for each cluster employees is solved, then a task assignment problem is solved for the resulted shifts. In the iterative procedure, they start by solving a minimized version of the tour-scheduling problem: without including the task assignment and the workstation group transition restriction. If the workstation group transition constraints are violated in the resulted solution, the demand is modified and the tour-scheduling problem is resolved again. Shifts are finally assigned to the employees. The iterative procedure shows better results, where nearly optimal solutions are achieved for instances with hundreds of employees and about five workstation groups, in around thirty minutes.

Sabar et al. [65] present a formal mathematical programming model for the employee task assignment in a multi-product sequential assembly line center. The model respects the employee task preferences and aims at minimizing the employee transitions between tasks. The model is validated using a small dataset and a commercial solver.

Hojati and Patil [38] present a linear programming based heuristic for scheduling tasks and shifts to part-time employees. The problem is decomposed into two main sub-problems: determining shifts and assigning shifts to employees. For the first part, a daily anonymous scheduling problem is solved for each task. For the shift assignment sub-problem, solving an integer program was not efficient in terms of computation time, thus an integer programming based heuristic is developed. The heuristic iteratively optimizes each employee schedule.

Lequy et al. [50] provides three integer programming models for assigning interruptible activities to the employee shifts. A mathematical programming based heuristic is presented to solve these models. Jin [41] studies the same problem but with both interruptible and uninterruptible tasks. A two-phase approach is developed. In the first phase, uninterruptible tasks are assigned to shifts, then the interruptible tasks are assigned to shifts at the second phase.

Munezero [57] deals with the same problem we are solving, the employee scheduling problem with inter-department transfers, with minor differences within some constraints. In the thesis, a two-phase heuristic is presented. For the first phase, a department decomposition is applied, then each department employee schedules are optimized using an integer linear program. In the second phase, under-covered time periods within all departments are fulfilled by transferring any available and qualified employee from another departments. A greedy algorithm and a formal integer program are presented for the second phase. The integer program showed better results than the greedy algorithm

2.4.1 Large neighborhood search for the employee scheduling problem

Dahmen and Rekik [20] solve the multi-activity, multi-day employee scheduling problem using a hybrid heuristic. They present a tabu search with large neighborhoods. The large neighborhoods are evaluated with a branch-and-bound procedure. The tabu search neighborhood is defined as follows: for the current solution, the day and the activity with the largest under-coverage are identified, all possible employee shifts for this activity during this day are enumerated. For the remaining days and activities, only shifts already existing in the current solution are enumerated, this is a worst destroy operator. Branch-and-bound solves a restricted set covering model for the employee scheduling problem using only these enumerated shifts, resulting in an optimal solution serving as the best neighbor for the current solution. Intensification and diversification are further used to escape from local optima. Good solutions are obtained using this hybrid heuristic. While the presented computational time seems to be high, the authors point to the heuristic ability to achieve good solutions for large instances.

Quimper and Rousseau [62] present a large neighborhood search for the multi-activity multi-skill employee scheduling problem. Formal language is used to model the problem: a grammar graph representing the problem constraints is created, a path in the graph represents a feasible sequence and a schedule is a set of feasible sequences. The large neighborhood search

destroy operator destroys a sequence from the current schedule and the repair operator replaces it by another sequence with lower cost. The heuristic achieves near optimal solutions with fast computational times compared to the exact mixed integer program solution.

2.4.2 Parallel metaheuristic for the employee scheduling problem

Martin et al. [53] solve the fair nurse roster problem using a parallel metaheuristic. Three metaheuristics are implemented: tabu, simulated annealing and variable neighborhood search. Each metaheuristic is called an agent and optimizes a variant of the objective function. Different neighborhood structures are used. A launcher first reads the problem, creates an initial solution and sends it to the different agents. Agents use an iterative cooperation protocol as follows: they start optimizing their current solution for a number of iterations. An *initiator* agent is chosen, it sends its best solution to all agents. Each agent captures good patterns from this solution and rebuilds a new starting solution before restarting the optimization for another set of iterations. The authors use the parallelism not only to accelerate the search or to seek better local optima, but they use parallelism to discover the best objective function that better models fairness within soft constraints violation. Results show that the use of the cooperative agents gives better results than using a standalone metaheuristic search, and the use of different objective functions helped the model to produce the fairest nurse roster.

CHAPTER 3 GENERAL ORGANIZATION OF THE THESIS

This thesis focuses on two variants of the heterogeneous employee scheduling problem, namely, the employee scheduling problem with inter-department transfers and the multi-job employee scheduling problem. While the literature is rich with works optimizing the ESP, no scalable solution method for the ESP-IDT has been proposed, and a matheuristic has never been applied and tested for the MJ-ESP. Furthermore, parallelism has not been exploited for both problems. This thesis addresses these research gaps by developing decomposition heuristics which use parallelism to solve the large-scale ESP-IDT and a parallel matheuristic for the MJ-ESP.

In general, this research project integrates decomposition heuristics and exact solution methods for optimizing the ESP variants, meanwhile, parallelization is maintained for all heuristics. Parallel computation helped to reduce the computation time, maintained heuristics scalability and robustness. The thesis is divided into three main parts. The first part in Chapter 4 corresponds to the article "A decomposition-based heuristic for large employee scheduling problems with inter-department transfers". The proposed decomposition heuristic transforms the intractable large problem instances into manageable ones, by introducing a novel parallel mono-department employee scheduling problem with derived inter-department transfers. This ESP-DIDT computational time is further reduced in Chapter 5, where a hybrid heuristic controlling the algorithms computation time is proposed. Chapter 6 addresses the MJ-ESP with a domain decomposition parallel large neighborhood search heuristic. The LNS iteratively destroys disjoint sub-schedules and repairs them in parallel with the commercial solver WFC, until the stopping criterion is met.

Chapter 7 generally discusses the research objectives and results, where solution methods from the literature along with research gaps are detailed. Chapter 8 is divided into three sections, the first starts by summarizing the presented work. Then a limitation analysis for every part of the thesis is presented in the second section. Finally, in the last section, interesting future work are suggested.

CHAPTER 4 ARTICLE 1 : A DECOMPOSITION-BASED HEURISTIC FOR LARGE EMPLOYEE SCHEDULING PROBLEMS WITH INTER-DEPARTMENT TRANSFERS

Dalia Attia[a,c], **Reinhard Bürgy**[b,c], **Guy Desaulniers** [a,c], **François Soumis** [a,c]

EURO Journal on Computational Optimization volume 7, pages325–357(2019)

[a] *Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Montréal (Québec), Canada*

[b] *Département d'Informatique, Université de Fribourg, Fribourg, Switzerland*

[c] *GERAD, Montréal (Québec), Canada*

Abstract: We consider a personalized employee scheduling problem with characteristics present in retail stores consisting of multiple departments. In the setting under study, each department generally covers its demand in employees over the planning horizon of a week by assigning shifts to its own staff. However, the employees can also be transferred to other departments under certain conditions for executing entire shifts or parts of shifts there. The transfer feature enables to improve the overall schedule quality considerably when compared to the non-transfer case. Given the complexity of the problem, we propose a three-phased decomposition-based heuristic. In the first phase, we consider each department separately and solve a simplified version of the mono-department scheduling problems. From the obtained solutions, we deduce inter-department shifts that could potentially reduce the overall cost. This is examined in the second phase by re-solving the scheduling problem of the first phase where the deduced inter-department shifts are included. In this phase, however, we decompose the scheduling problem by time, looking at each day separately. From the obtained schedules, we then devise inter-department demand curves, which specify the number of transfers between departments over time. In the third phase, we decompose the initial scheduling problem into mono-department problems using these inter-department demand curves. Consequently, our approach makes it possible to solve mono-department optimization problems to get an overall schedule while still benefiting from the employee transfer feature. In all three phases, the scheduling problems are formulated as mixed-integer linear programs. We show through extensive computational experiments on instances with up to

25 departments and 1000 employees that the method provides high-quality solutions within reasonable computation times.

Keywords: Employees scheduling, shift scheduling, multi-department, retail industry, heterogeneous workforce, mixed-integer linear programming, decomposition

Acknowledgments: This work was funded by Kronos Canadian Systems and the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant RDCPJ 468716 - 14. The financial support is greatly appreciated.

4.1 Introduction

Personnel scheduling consists of assigning employees to activities over time. As explained in Thompson [72], managers of all types of organizations should care about this recurrent task for the following three primary reasons. First, personnel scheduling has a direct consequence on the profitability. An under-supply of employees presumably leads to poor customer service which translates into lost business, while an over-supply is related to an excess of salary costs. Second, employees have specific work preferences, for example, with respect to the assigned activities, the time of day they work, the length of their shifts, and with whom they work. Meeting these preferences generally translates to better work satisfaction and performance. Third, managers usually dedicate a significant amount of time and effort for developing employee schedules as it is no easy task to construct admissible, high-quality solutions. A (semi-) automated scheduling process frees up precious time, so that the managers can spend more time for other important tasks.

Researchers in management science and operations research have established a valuable body of knowledge over the last decades helping the managers to better handle the labor scheduling task. In particular, scientists have introduced mathematical models capturing the underlying decision-making problems, developed methods for solving these inherently difficult optimization models; and studied applications of the developed models and methods in industry underpinning the value of the developments. An important aspect of this research field is that no generally applicable employee scheduling model and method exists. On the contrary, the specific characteristics on the industry and company levels make it necessary to establish specialized solutions [33]. As a consequence, employee scheduling problems have been studied extensively in various domains, including hospitals [76, 49], air transport [29, 43], factories [11, 34], restaurants [52, 39], and retail stores [42, 12]. We refer the reader to the

survey papers of Ernst et al. [33], Van den Bergh et al. [73], and De Bruecker et al. [24] for a complete overview of the employee scheduling research field.

In this paper, we address a scheduling problem typically arising in the retail industry and call it the employee scheduling problem with inter-department transfers (ESP-IDT). It consists of scheduling the employees of a store during a multi-day time horizon. While we only consider weekly problems, (slightly) longer time horizons can be considered without major changes. The given time horizon is split into small consecutive time periods of a fixed duration, for example 15 minutes. We assume that the store is partitioned into departments, which are units within the store with some degrees of independence. Each department has its own (internal) employees, of which some are qualified to work in other (external) departments, too. To cover its demand in employees, a department can use its own staff or can borrow qualified employees from other departments as required. The transfer of employees between departments is, however, regulated as these transfers may lead to some inconvenience on the employees' side, and to higher managerial complexity and some loss of productivity on the employer's side. More specifically, the employees work in shifts, and a shift is either fully executed in the home department of the assigned employee or in one of the other departments he/she is qualified for. In addition, we also consider shifts where, after a certain time, the employee is transferred from one department to another. In this case, both work blocks must not be too short and one of the two involved departments must be the employee's home department.

We also consider the following standard characteristics of personalized employee scheduling problems. We assume that the days-off planning has been established telling on which days each employee can be assigned to work a shift. Shift profiles can be specified to limit the possible start periods and lengths of the shifts, and the rest period between two consecutive shifts of an employee must not be too short. We assume that breaks within shifts are assigned in real-time depending on the observed demand and do not model them explicitly. The demand forecasts may be increased in certain time periods to account for the breaks in the scheduling problem. To measure the quality of the schedules, we capture demand under- and over-coverage costs, salary costs, and transfer costs, which is a penalty for the time employees spend in external departments. We seek to find an employee schedule with minimum total cost.

The specific features of ESP-IDT are the simultaneous consideration of multiple departments, each having its own employees, and the possibility to transfer employees between departments. Multi-department problems occur, for example, in large department stores, in furniture retailers, and in supermarkets. In these working environments, employees are

typically assigned to one department but have the knowledge and qualifications to work in other departments, too. This flexibility is valuable for the employee scheduling task as it enables to better match supply and demand in employees. Motivated by current practice, we only consider specific transfer shifts and assign some costs to the time executed in external departments to handle the negative consequences of transfers mentioned before. From an optimization perspective, it is not easy to make use of the transfers as, first, there is a large number of possible inter-department shifts to consider, and second, there exists no natural decomposition into mono-department problems when considering inter-department transfers. Hence, the overall optimization problem tends to be extremely large and difficult to solve.

In this paper, we contribute to the employee scheduling research field by proposing a three-phased decomposition-based heuristic for solving ESP-IDT. In the first phase, we consider each department separately and solve a simpler version of the initial problem obtained by omitting individual features of the employees. From the obtained schedules, we deduce inter-department shifts that potentially lead to schedule improvements. In the second phase, we include these inter-department shifts to the optimization problem of the first phase. This time, we decompose the problem not by department but by time and solve the so-obtained daily scheduling problems. From the obtained schedules, we then devise inter-department demand curves, specifying the number of transfers between departments for each time period. In the third phase, we decompose the initial scheduling problem into mono-department problems using these inter-department demand curves. In all three phases, the optimization problems are formulated as mixed-integer linear programs (MILPs) and solved with a commercial MILP solver.

The remaining part of this paper is organized as follows. The next section provides a brief literature review pointing to some related works. Section 4.3 describes ESP-IDT formally, provides a MILP formulation, and introduces an illustrative example. In Section 4.4, we propose a three-phase solution method for ESP-IDT and illustrate it with our example. Section 4.5 describes the extensive computational experiments that are executed to evaluate the method developed in the previous section. A conclusion is provided in Section 4.6, and the appendix contains the detailed numerical results of the experiments.

4.2 Literature

As one can see in the survey papers listed in the previous section, employee scheduling problems or related questions are addressed in a huge number of scientific works. In this section, we only point to the articles that are, in our view, the most relevant with respect to our study.

A number of articles, including Loucks and Jacobs [51], Sabar et al. [65**?**], Côté et al. [16], Dahmen and Rekik [20], consider personnel scheduling problems with multiple activities and multi-activity shifts. In this setting, the goal is not only to specify the work and rest hours for each employee, but also to define the activities assigned to the work periods, where typically qualifications and possibly preferences of the employees must be respected. These problems are closely related to our study since a department can be seen as a specific activity. Similar as in our work, the transition from one activity to another within a shift is usually regulated by, for example, imposing minimum work durations before switching to another activity. However, the notion of main activity, which would reflect the home department in our study, is usually not present. This is an important difference as we penalize the time spent in external departments, reflecting the wish to assign employees to their home department and only using the transfers as needed. This also limits the possible downside effects of transfers, which are, for example, increased managerial complexity, higher dissatisfaction of the employees, and some loss of productivity. We also remark that the number of employees is typically much larger in multi-department scheduling problems than in multi-activity environments. It is, for instance, not unusual that up to 1000 employees work in multi-department stores while a single department may consist of about 100 employees.

The home department feature is, to a certain extent, considered in Bard and Wan [9]. Their goal is to select an optimal size and composition of full-time and part-time employees when the demand in employees is specified by workstation group (WSG), which is a department in our terminology. The employees must be assigned to a home WSG but can also work for other WSGs under some pre-defined conditions. They highlight that the transfer feature is necessary to avoid excess idle time. However, the number of transfers between WSGs are to be kept small due to layout restrictions, union agreements, and supervisory preferences. The authors develop two versions of a multi-stage approach to solve their problem, and test them on data provided by a U.S. Postal Service mail processing and distribution center. This work clearly confirms the value and difficulty of including employee transfers between departments. Their context is, however, structurally different from ours. We consider a setting where the workforce is given and employees have individual preferences and qualifications, while Bard and Wan's goal is to optimize the workforce. Furthermore, ESP-IDT specifies less restrictive shift profile rules, which leads to a substantially larger set of feasible shifts.

Departments are also considered in the study of Bard and Purnomo [7], where hospital-wide reactive scheduling of nurses is considered. More specifically, each unit, which corresponds to a department in our terminology, establishes a midterm schedule independently. The units try to cover their estimated demand with their own nurses in the best possible way. Bard and Purnomo consider these schedules as input, and address the problem of reactively

adjusting the nurses' work schedules of the next 24 hours to account for the daily fluctuations in the supply and demand of nurses. One of the alternatives for improving the schedule is the transfer of nurses from their home unit to other units as needed. The authors develop a specialized branch-and-price algorithm that solves instances with up to 200 nurses within ten minutes to optimality. While the notion of home department is also important in this study, the general setting is clearly different from ESP-IDT. First of all, they consider a reactive scheduling problem while ESP-IDT considers the initial schedule generation process, and second, in contrast to the retail industry, where the shifts start and end at many different time points each day, there are only few alternatives when looking at nurse shifts.

The most closely related work to our study is Dahmen et al. [22]. They consider ESP-IDT except that they enforce the assignment of a shift during a non-day-off. Indeed, the authors consider a multi-department employee scheduling problem with a weekly time horizon, the employees have a home department and qualifications to work in other departments, and costs are assigned for over-coverage and under-coverage of the department demands and for transfers and work times. Structurally, a main difference is that a shift must be executed during a non-day-off of an employee in their study, while we only state that the given days-off of the employees must be respected and a shift may or may not be assigned to an employee for all other days. Having this choice is particularly important with part-time employees. For those, one typically specifies the days during which they are not available, and all other days can be chosen as workdays. Dahmen et al. propose a two-stage decomposition heuristic for solving their scheduling problem. In the first stage, they solve a smaller optimization problem where the data is aggregated and transfers are somewhat approximated. In the second stage, they generate optimized schedules by solving a MILP, in which a subset of promising shifts, derived from the outputs of the first stage, is present. While the "days-off" difference between their and our study has some effects on the structure of feasible schedules, we show that the method we propose for ESP-IDT can also be applied to the setting of Dahmen et al. [22] with minor changes.

4.3 The employee scheduling problem with inter-department transfers

In this section, we first define ESP-IDT formally, then formulate it as a mixed-integer program, and finally introduce an illustrative example that will be used throughout Section 4.4.

4.3.1 Problem statement

We consider a planning horizon of one week given by days J_1 to J_7. The time horizon is divided into consecutive short time periods of some predefined length (for example, 15 minutes). Let $P = \{p_1, \ldots, p_{|P|}\}$ be an ordered set comprising the resulting time periods, where p_r refers to the rth period of the week. We typically use p (without an index) for a generic period in P. Each period $p \in P$ starts and ends on a specific day of the week $\mathrm{DAY}(p) \in \{J_1, \ldots, J_7\}$. In the sequel, time lengths will generally be specified as units of time periods.

ESP-IDT involves a set D of departments. A target demand b_{pd} in employees needs to be satisfied for each department $d \in D$ at each time period $p \in P$. We consider the complete planning horizon, so that a possible closing time of all departments simply results in zero demand for all departments in all the corresponding time periods. We allow for under- and over-coverage of this demand but penalize both deviations in the objective function with costs linear in the size of the deviation. Denote by c_d^{un} and c_d^{ov} the unit penalty cost paid for under- and over-coverage, respectively, in department $d \in D$. We abstain from establishing unit penalty costs that depend on the time period to keep the notation slightly simpler.

The demands can be covered by a set E of employees. Each employee e has a specific home department $d_e^{\mathrm{h}} \in D$ and is qualified to work in a given set of departments $D_e \subseteq D$. Clearly, $d_e^{\mathrm{h}} \in D_e$ holds. For each department $d \in D$, denote by $E_d \subseteq E$ the set of employees with d as home department. To capture the employees' work time costs, a unit cost of c^{wt} is charged for each period an employee is working, and additionally, a unit transfer cost of c^{tr} is charged for each period an employee is working in another department than his/her home department.

We assume that the days-off planning has already been established, so that each employee $e \in E$ has a predefined set of work days, say $J(e)$, and rest days over the planning horizon. The employees work in shifts. A shift s is defined by an employee $\mathrm{EMP}(s)$, a start period $\mathrm{STA}(s) \in P$, an end period $\mathrm{END}(s) \in P$ as well as the department to which the employee is assigned in each period covered by this shift. A shift may start at one day and finish at the next, in which case the shift is considered to be assigned on the day of the starting period. Let $P(s, d)$ be the set of periods employee $\mathrm{EMP}(s)$ works in department d during shift s and denote by $P(s) = \bigcup_{d \in D} P(s, d)$ the complete set of periods covered in shift s. A shift is called *internal* or *external* if it is completely executed in the assigned employee's home department or in an external department, respectively. If more than one department is associated with a shift, then it is called a *transfer* shift. The cost c_s to execute shift s can be inferred from the work time and transfer costs, i.e., $c_s = c^{\mathrm{wt}}|P(s)| + c^{\mathrm{tr}}(|P(s)| - |P(s, d_e^{\mathrm{h}})|)$. Shift profiles

can be specified to restrict both the start periods and the total lengths of the shifts. For example, one can state in shift profiles that each shift must start at a full hour and its length must be a multiple of an hour. Shift profile definitions are common in practice. They enable, for example, to reduce the managerial complexity.

We only consider shifts fulfilling the following constraints. A shift s is either fully executed in one of the departments D_e the assigned employee $e = \text{EMP}(s)$ is qualified for, or after executing a first work block in one of the departments in D_e, the employee e is transferred to another department in D_e to execute the second work block. In this case, however, one of the assigned departments must be the home department d_e^{h}, and both work block lengths must be at least of a given minimum duration. Furthermore, shift s must start during a work day of e, i.e., $\text{DAY}(\text{STA}(s)) \in J(e)$, and must satisfy the shift profile rules mentioned before.

Denote by $\mathcal{S}$ the complete set of feasible shifts. For each employee $e \in E$, let $\mathcal{S}_e = \{e \in \mathcal{S} : e = \text{EMP}(s)\}$ be the set of feasible shifts s of employee e. An employee schedule is obtained by selecting a set of shifts from $\mathcal{S}$ that must be executed. We call a schedule $S \subseteq \mathcal{S}$ *feasible* if each employee $e \in E$ is assigned to at most one shift per day in $J(e)$, works at most $t_e^{\max}$ periods over the planning horizon, and a minimum number of rest periods $r^{\min}$ between consecutive shifts of e in schedule S is respected.

For any schedule $S \subseteq \mathcal{S}$, define $\text{COV}(S, p, d) = |\{s \in S : p \in P(s, d)\}|$ as the number of employees present in department $d \in D$ during period $p \in P$. The cost $c(S)$ of a schedule $S \subseteq \mathcal{S}$ is given by the sum of its work time and transfer costs captured by the shift costs and its demand under-coverage and over-coverage costs:

$$c(S) = \sum_{s \in S} c_s + \sum_{p \in P} \sum_{d \in D} \left[c_d^{\text{un}} (b_{pd} - \text{COV}(S, p, d))^+ + c_d^{\text{ov}} (\text{COV}(S, p, d) - b_{pd})^+ \right] \tag{4.1}$$

where notation $(z)^+$ is a shortcut for $\max(0, z)$.

ESP-IDT consists of finding a feasible schedule $S \subseteq \mathcal{S}$ with minimum cost $c(S)$.

4.3.2 A mixed-integer programming formulation

We now develop a MILP for ESP-IDT. For each shift $s \in \mathcal{S}$, introduce a binary variable x_s taking value 1 if s is selected and 0, otherwise. To capture the over- and under-coverage of the demand, introduce two non-negative variables y_{pd}^- and y_{pd}^+ for each period $p \in P$ and

department $d \in D$. Then, the following MILP describes ESP-IDT:

$$\text{Minimize} \sum_{s \in \mathcal{S}} c_s x_s + \sum_{p \in P} \sum_{d \in D} (c_d^{\text{un}} y_{pd}^- + c_d^{\text{ov}} y_{pd}^+) \tag{4.2a}$$

subject to

$$\sum_{\substack{s \in \mathcal{S}: \\ p \in P(s,d)}} x_s - y_{pd}^+ + y_{pd}^- = b_{pd} \qquad \text{for all } p \in P \text{ and } d \in D, \tag{4.2b}$$

$$\sum_{\substack{s \in \mathcal{S}_e: \\ \text{DAY}(\text{STA}(s))=j}} x_s \leq 1 \qquad \text{for all } e \in E \text{ and } j \in J(e), \tag{4.2c}$$

$$\sum_{s \in \mathcal{S}_e} |P(s)| x_s \leq t_e^{\max} \qquad \text{for all } e \in E, \tag{4.2d}$$

$$\sum_{\substack{s \in \mathcal{S}_e: \\ \{p_k, \ldots, p_{k+r^{\min}}\} \cap P(s) \neq \emptyset}} x_s \leq 1 \qquad \text{for all } e \in E \text{ and } k \in \{1, \ldots, |P| - r^{\min}\}, \tag{4.2e}$$

$$x_s \in \{0, 1\} \qquad \text{for all } s \in \mathcal{S}, \tag{4.2f}$$

$$y_{pd}^-,\ y_{pd}^+ \geq 0 \qquad \text{for all } p \in P \text{ and } d \in D. \tag{4.2g}$$

The objective function (4.2a) minimizes the total cost as defined in (4.1). Constraints (4.2b) link the variables y_{pd}^- and y_{pd}^+ to the variables x_s according to their meaning. Constraints (4.2c) limit the number of shifts assigned to an employee for each of his/her potential workdays. Constraints (4.2d) ensure that the employees' maximum weekly work time limits are respected. Constraints (4.2e) model the minimum rest time of $r^{\min}$ periods between two shifts for each employee e by imposing that for each set of $r^{\min} + 1$ consecutive periods, at most one selected shift of e must intersect with them. This ensures that, after the end of a shift of e, the next must start at least $r^{\min}$ periods later. Finally, constraints (4.2f) and (4.2g) specify the domains of the decision variables. Note that using constraints (4.2c), one can eliminate some of the constraints (4.2e). Indeed, if for some k, the set of time periods $\{p_k, \ldots, p_{k+r^{\min}}\}$ all belong to the same day, then the corresponding constraint in (4.2e) can be dropped.

The number of variables in (4.2) and the difficulty to solve it mainly depends on the number of shifts. This, in turn, depends on various factors such as the number of departments, the number of employees and their qualifications, the work block length constraints and the shift profiles. The number of shifts is typically in the order of multiple millions even in instances for medium-sized organizations with, for example, five departments, 200 employees working five days a week in shifts with a length between five to eight hours. In particular, the possibility to transfer employees between departments drastically increases the scheduling flexibility and

the number of shifts to be considered. We refer the reader to Section 4.5 for some specific sizes of shift sets. Solving (4.2) for larger instances is simply impractical with state-of-the-art MILP solvers. Indeed, not only the optimization process but already the task of generating and loading the input data may pose some major problems.

4.3.3 An example

We introduce a small example to be used in Section 4.4 for illustrating our method. In this example, the time horizon is divided in periods with a length of two hours (which would typically be too long in practice), so that there are 84 periods in total. Figure 4.1 describes the two departments of the example by showing their demand curves and providing data about the employees. Each employee is listed in his/her home department. The name of an employee is surrounded by a rectangle if he/she can work in the other department, too. The days-off are indicated by a black line at the vertical position of the corresponding periods. The shift profiles specify that the minimum work block length is one period, the shift length must be between 2 and 4 periods, and there is no restriction with respect to the start period. The maximum total work time of each employee is 20 periods. The unit under- and over-coverage cost (per employee and period) is 18.8 and 9.4, respectively, the unit work time cost is 0.3, and the unit transfer cost is 0.2.

4.4 A three-phase solution method for ESP-IDT

When dealing with large multi-department employee scheduling problems, a typical approach is to decompose the overall problem into mono-department problems that are then solved separately. For ESP-IDT, the decomposition into mono-department problems is not obvious. When neglecting the chance to transfer employees to other departments, this decomposition is naturally obtained. However, as shown in the literature and by our computational results in Section 4.5, including the transfer possibility leads to substantially improved schedules. We will therefore present a method that makes it possible to solve mono-department optimization problems to get an overall schedule while still benefiting from the employee transfer feature.

As depicted in Figure 4.2, our method, called multi-phase decomposition heuristic (MP-DH), is composed of three phases, which can briefly be described as follows. In the first phase, we look at each department separately and find an optimized schedule using only internal shifts. To reduce the computational effort, a simplified version of model (4.2) is used for this purpose in which employees are not considered individually resulting in a so-called anonymous employee scheduling problem. Based on the obtained schedules, we then generate

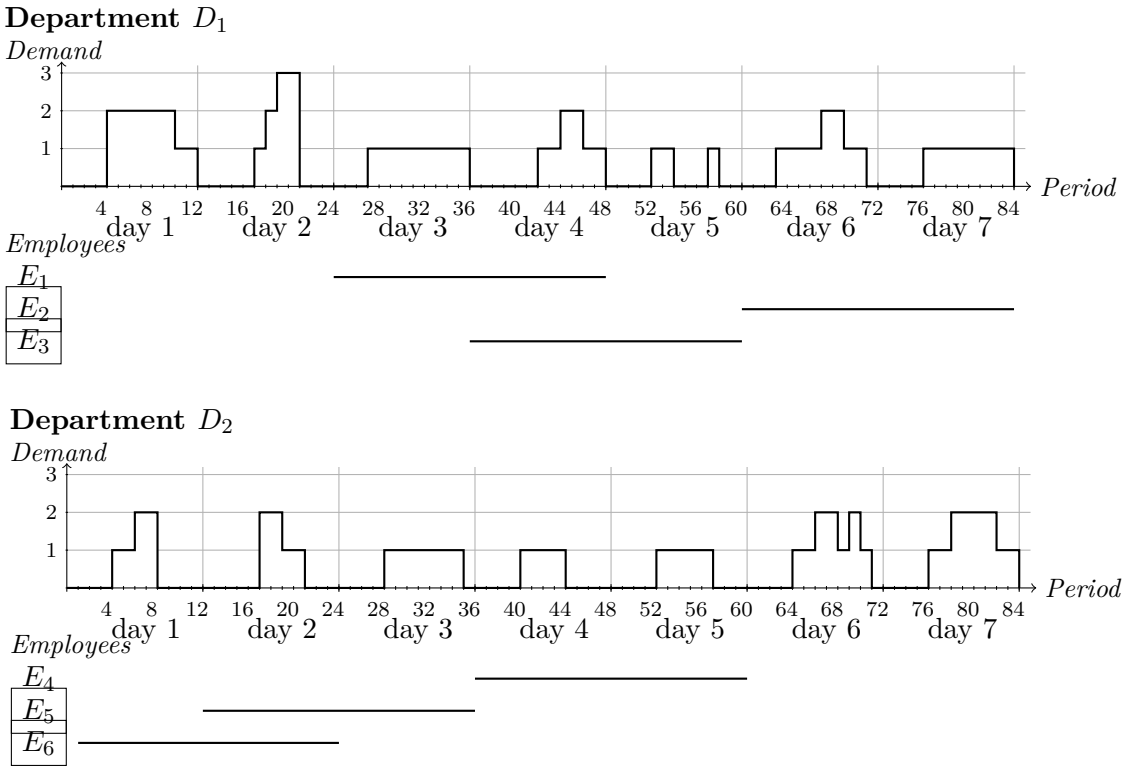

Figure 4.1 The two departments of our illustrative example.

external and transfer shifts that can possibly improve these schedules. In the second phase, the anonymous employee scheduling problem used in the first phase is re-solved using the external and transfer shifts generated in the first phase. This time, we do not decompose the problem into mono-department problems as we aim to evaluate the impact of the employee transfers. However, to keep the computational burden reasonable, we decompose the problem by time, looking at each day separately. The solutions of these daily, anonymous employee scheduling problems give rise to inter-department demand curves, i.e., for each time period, we decide how many employees of one department must be serving in another department either within external or transfer shifts. Based on these demand curves, we then solve a version of the global model (4.2) in the third phase. The inter-department demand curves make it possible to decompose the problem into mono-department problems and to reasonably limit the number of shifts that are considered. The next sections describe the three phases in detail.

4.4.1 First phase: Generate promising external and transfer shifts

In this phase of MP-DH, we generate promising external and transfer shifts using the following three steps. In a first step, we determine periods where the departments will possibly not be able to match their demands perfectly with internal shifts. Periods with demand under-coverage are formidable candidates to be covered by external employees. Similarly, periods that indirectly caused demand over-coverage in other periods are also good candidates for a coverage by external employees. We extract these critical periods in a second step and use them to create a set of external and transfer shifts in a third step.

Determine under- and over-coverage curves

The periods with possible demand under- and over-coverage are determined with a simplified MILP of (4.2). We not only consider each department separately, but also deal with an anonymous version of the scheduling problem, i.e., the employees are not considered individually.

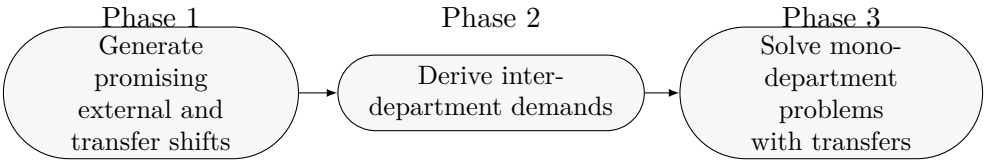

Figure 4.2 The three phases of MP-DH.

More formally, for each department $d \in D$, we derive the following model from (4.2). First, we only consider internal shifts and do not associate specific employees with a shift. Hence, for a given department d, a shift s is fully specified by its start and end periods. Note that the cost c_s of such an anonymous shift is still well-defined. Denote by $\mathcal{S}_d^{\text{dep}}$ the complete set of shifts for department d. The size of this set is small. Indeed, suppose that the shift starting periods are restricted to full hours, then at most $24 \cdot 7 = 168$ starting periods per week exist. If five possible shift lengths, for example four, five, six, seven, and eight hours, are considered, then no more than 840 shifts are obtained. When considering anonymous shifts, some employee specific constraints cannot be modeled in the same manner, such as the employees' maximum of one shift per workday, maximum work time per week, and the minimum rest periods. We replace some of these constraints by more aggregated versions.

Specifically, introduce a non-negative integer variable x_s for each shift $s \in \mathcal{S}_d^{\text{dep}}$ indicating how many employees work shift s and two non-negative variables y_{pd}^- and y_{pd}^+ for each period $p \in P$ capturing the under- and over-coverage of the demand. Then, solve the following MILP for department $d \in D$:

$$\text{Minimize} \quad \sum_{s \in \mathcal{S}_d^{\text{dep}}} c_s x_s + \sum_{p \in P} (c_d^{\text{un}} y_{pd}^- + c_d^{\text{ov}} y_{pd}^+) \tag{4.3a}$$

subject to

$$\sum_{\substack{s \in \mathcal{S}_d^{\text{dep}}: \\ p \in P(s)}} x_s - y_{pd}^+ + y_{pd}^- = b_{pd} \qquad \text{for all } p \in P, \tag{4.3b}$$

$$\sum_{\substack{s \in \mathcal{S}_d^{\text{dep}}: \\ \text{DAY}(\text{STA}(s)) = J_r}} x_s \leq |\{e \in E_d : J_r \in J(e)\}| \qquad \text{for all } r \in \{1, \ldots, 7\}, \tag{4.3c}$$

$$\sum_{s \in \mathcal{S}_d^{\text{dep}}} |P(s)|\, x_s \leq t_e^{\max} |E_d|, \tag{4.3d}$$

$$x_s \in \mathbb{Z}_{\geq 0} \qquad \text{for all } s \in \mathcal{S}_d^{\text{dep}}, \tag{4.3e}$$

$$y_{pd}^-,\, y_{pd}^+ \geq 0 \qquad \text{for all } p \in P. \tag{4.3f}$$

The objective function (4.3a) captures the shift costs and the demand under- and over-coverage costs of department d as in (4.2a). Constraints (4.3b) link the variables y_{pd}^- and y_{pd}^+ to the variables x_s as in (4.2b). For each day, a constraint in (4.3c) limits the total number of shifts starting at this day to the number of employees available at day J_r in department d. These constraints represent the restriction given in (4.2c). Observing that the right-hand

side of constraint (4.3d) is an upper bound on the total time available of all employees of department d over the planning horizon, (4.3d) ensures that this upper bound is respected, reflecting constraints (4.2d). Finally, constraints (4.3e) and (4.3f) specify the domains of the decision variables.

For our example, Figure 4.3 shows the 30 shifts selected by solving model (4.3) and the resulting available capacity in employees (gray area under the demand curves). For department D_1, we observe that there is an under-coverage of one employee in periods 5, 10, 28, 45 to 48, 64, and 69, and an over-coverage of one employee in period 59. For department D_2, there is an under-coverage of one employee in periods 18 to 19.

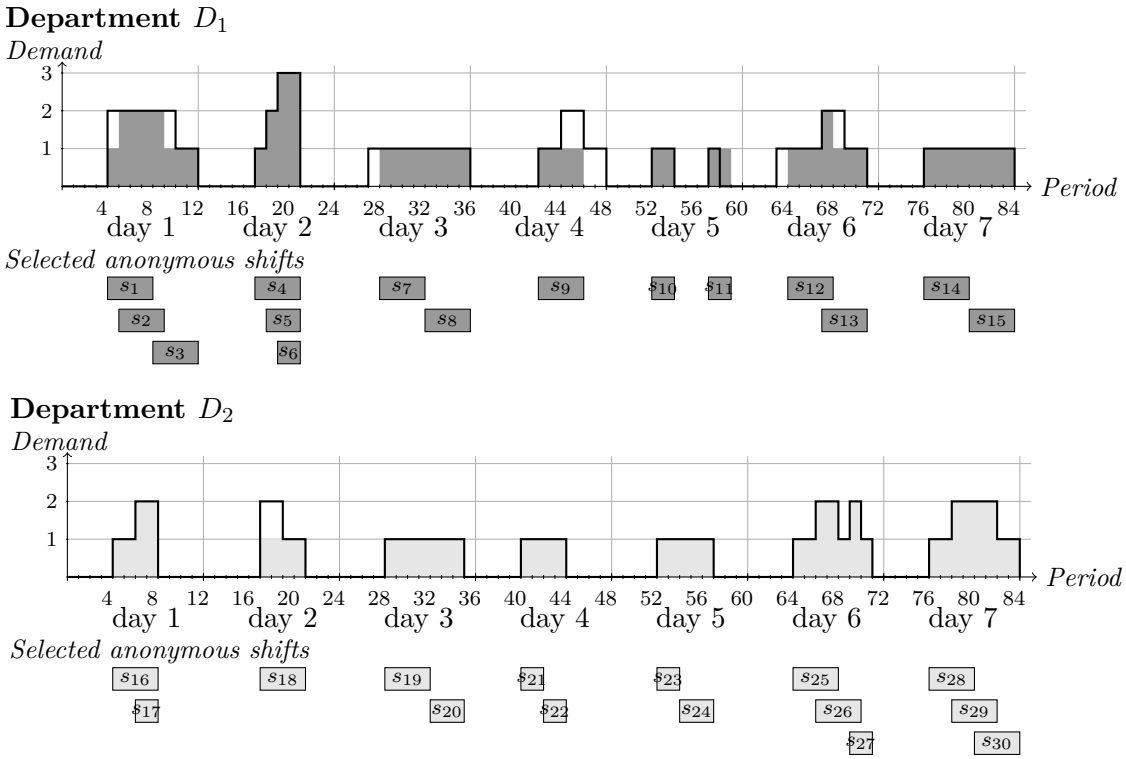

Figure 4.3 An optimal solution of model (4.3) for each department of our example.

Select critical time intervals

Periods with demand under-coverage and those implying demand over-coverage in other periods are considered as critical and are, in our view, good candidates to be covered with external and transfer shifts. We are especially interested in extracting critical time intervals given by consecutive critical periods. Denote a time interval $i = [\text{STA}(i), \text{END}(i)]$ by its start period $\text{STA}(i)$ and its end period $\text{END}(i)$.

For each department $d \in D$, we extract a set I_d^{crit} of critical time intervals from demand under-coverage and over-coverage of the solution of model (4.3) as follows.

For the demand under-coverage $y_{pd}^-, p \in P$, we use the simple scanning procedure described in Algorithm 1 to extract critical time intervals. By scanning through all periods, it finds maximal intervals with under-coverage and stores them in the set of critical time intervals. Then, the under-coverage is reduced by one in all periods and the previous scanning step is repeated. In our example, this procedure generates the critical intervals $[p_5, p_5]$, $[p_{10}, p_{10}]$, $[p_{28}, p_{28}]$, $[p_{45}, p_{48}]$, $[p_{64}, p_{64}]$, and $[p_{69}, p_{69}]$ for department D_1 and $[p_{18}, p_{19}]$ for department D_2 from the solution depicted in Figure 4.3.

As demand over-coverage is typically less costly (per unit) than is under-coverage, we are only interested in over-coverage intervals containing at least a given number of consecutive periods γ (for example, the number of periods that corresponds to one hour). With this parameter, we then apply Algorithm 2 to generate additional critical time intervals for set I_d^{crit}. The reasoning behind this algorithm is the following. As shift s was chosen to be executed, it seems not to be promising to remove s from the schedule because a (costlier) under-coverage would occur in the periods between the start of s and the start of i or between the end of i and the end of s. Both situations are reflected in the created critical intervals i' and i''. These intervals allow us then to transfer some of the demand in these intervals to employees from other departments. In our example, there is an over-coverage in department D_1 in period p_{59}. This over-coverage is caused by the shift starting in period p_{58} and ending in p_{59}. We therefore add the critical interval $[p_{58}, p_{58}]$ for department D_1.

The solution provided by model (4.3) is one of typically many optimal solutions, and other optimal solutions may possess different demand under- and over-coverage curves, which would result in different critical time intervals. To partially cope with this ambiguity, we try to first generate other optimal solutions by pre- or postponing single shifts (without changing the shift lengths) and then extract critical intervals with the steps described above. Algorithm 3 describes the version with preponing a single shift in detail. The version with postponing a shift is simply obtained from Algorithm 3 by increasing the start and end of the shift by one period in line 6 of Algorithm 3. Looking at our example in Figure 4.3, we get another optimal solution by starting shift s_2 one period earlier. With this solution, we generate a new critical interval $[p_9, p_9]$ caused by an under-coverage. Note that we refrain from listing all generated intervals in the example.

Finally, for any pair i and i' of time intervals in I_d^{crit}, if i' directly starts after the end of i, we add the concatenated interval i^* with $\text{STA}(i^*) = \text{STA}(i)$ and $\text{END}(i^*) = \text{END}(i')$ to set I_d^{crit}. The reasoning is that it could be beneficial to simultaneously consider the two critical

Algorithm 1: Extraction of critical time intervals from under-coverage.

while *there exists some period* p *with* $y^-_{pd} > 0$ **do**
  Start with $r = 0$.
  while $r < |P| + 1$ **do**
    Increase r by 1 until finding a period p_r with $y^-_{p_r,d} > 0$ or $r = |P| + 1$.
    if $r < |P| + 1$ **then**
      Create an interval i and set STA(i) to p_r.
      Continue to increase r by 1 until finding a period p_r with $y^-_{p_r,d} = 0$ or $r = |P| + 1$.
      Set END(i) to p_{r-1}, and add interval i to set I^{crit}_d.
  end
  Decrease the under-coverage by one in all periods, i.e., set y^-_{pd} to $\max(y^-_{pd} - 1, 0)$ for all $p \in P$.
end
Note: Some intervals may be generated multiple times. However, an interval is only added to set I^{crit}_d if it is not yet present in this set.

Algorithm 2: Extraction of critical time intervals from over-coverage.

`// First, extract a set of over-coverage intervals` I^{oc}`.`
Start with $r = 0$.
while $r < |P| + 1$ **do**
  Increase r by 1 until finding a period p_r with $y^+_{p_r,d} > 0$ or $r = |P| + 1$.
  if $r < |P| + 1$ **then**
    Create an interval i and set STA(i) to p_r.
    Continue to increase r by 1 until finding a period p_r with $y^+_{p_r,d} = 0$ or $r = |P| + 1$.
    Set END(i) to p_{r-1}
    if *length of* i *is at least* γ **then**
      Add interval i to set I^{oc}.
end
`// Then, generate critical intervals from the over-coverage time intervals.`
foreach $i = [STA(i), END(i)] \in I^{oc}$ **do**
  foreach *selected shift* s*, i.e., with* $x_s = 1$*, intersecting with interval* i **do**
    if *shift* s *starts before interval* i **then**
      Add $i' = [\text{STA}(s), p']$ to set I^{crit}_d, where p' is the period preceding STA(i).
    if *shift* s *ends after interval* i **then**
      Add $i'' = [p', \text{END}(s)]$ to set I^{crit}_d, where p' is the period succeeding END(i).
  end
end
Note: An interval is only added to set I^{crit}_d if it is not yet present in this set.

Algorithm 3: Generate alternative optimal solutions and derive critical intervals.

```
1  foreach department d ∈ D do
2      foreach selected shift s, i.e., with x_s = 1, in department d do
           // Variable altSol stores the generated alternative optimal solution.
3          altSol ← null
4          stopWhile ← false
5          while STA(s) ≠ p_1 and stopWhile = false do
6              decrease STA(s) and END(s) by one period;
7              if shift profile rules are fullfilled by s then
8                  if solution of department d with updated s is optimal then
9                      altSol ← solution with updated s.
10                 else
11                     stopWhile ← true
12                 end
13         end
14         if altSol is not null then
15             Apply Algorithms 1 and 2 with solution altSol.
16     end
17 end
```

time intervals. In our example, department D_1 has the critical intervals $[p_9, p_9]$ and $[p_{10}, p_{10}]$. Hence, we also add the concatenated interval $[p_9, p_{10}]$.

Create transfer and external shifts

In the last step of this phase, we create the anonymous transfer and external shifts that are considered in the next phase. For this purpose, we will characterize an external shift s by its start period $\text{STA}(s) \in P$, its end period $\text{STA}(s) \in P$, the department $D(s) \in D$ providing an employee to execute shift s, and the department where s is executed. For any anonymous transfer shift, we further specify a transfer period, after which the work place is changed and an indication whether the work block before or after the transfer period is executed in the external department.

For each department $d \in D$, we sequentially consider each of its critical intervals i in I_d^{crit} to generate the following shifts. We create all feasible (with respect to the pre-defined shift profile constraints, see Section 4.3.1) anonymous external shifts starting no more than δ time periods before $\text{STA}(i)$ and ending no more than δ time periods after $\text{END}(i)$. We do not impose that the start and end times of the created shifts coincide with interval i as it is then typically impossible to fulfill the shift profile constraints. The parameter δ should be set with this in mind. Similarly, we create all feasible transfer shifts whose first work block is in department d. In this case, we set the transfer period to $\text{END}(i)$. In the same manner, we

create all feasible transfer shifts whose second work block is in department d. In this case, we set the transfer period to the period preceding STA(i). The described external and transfer shifts are generated for each department of origin d' that has at least one employee qualified to work in (the external) department d. Denote by $\mathcal{S}_d^{\text{ex/tr}}$ the set of external and transfer shifts generated for department d.

Consider the critical interval $[p_{45}, p_{48}]$ of department D_1 in our example and let δ be one period. We create all external shifts of length between two and four periods starting at or after p_{44} and ending at or before p_{49}. Then, we generate the transfer shifts with a first work block in D_1 starting either at p_{46}, p_{47}, or p_{48} and ending at p_{48} and a second work block in D_2 starting at p_{49} and ending either at p_{49}, p_{50}, or p_{51}. Clearly, the total shift length must be at most four periods as specified in the shift profiles. Similarly, we create the transfer shifts with a first work block in D_2 starting either at p_{42}, p_{43}, or p_{44} and ending at p_{44} and a second work block in D_1 starting at p_{45} and ending either at p_{45}, p_{46}, or p_{47}. Again, both work blocks together must contain at most four periods.

We finally remark that the number of generated external and transfer shifts can become very large, particularly for instances where the demand in employees cannot be matched well only with internal shifts, which often occurs when the demand highly fluctuates over time. This is a problem since each generated shift will be a variable in one of the MILPs of the daily scheduling problems addressed in the next phase (see Section 4.4.2). Hence, to keep these MILPs reasonably small, we set a restriction for the total number of shifts (internal, external and transfer) starting at a same day. More specifically, if the number of shifts in $\mathcal{S}_d^{\text{dep}} \cup \mathcal{S}_d^{\text{ex/tr}}$ starting at some day d is larger than β, we delete all transfer and external shifts that were generated (fully or partially) due to demand over-coverage. Parameter β should be chosen so as the MILPs of the next phase can be solved within reasonable computation time.

4.4.2 Second phase: Derive inter-department demands

In the second phase of MP-DH, we first solve a daily, anonymous employee scheduling problem to determine how to best use the created anonymous external and transfer shifts to reduce the demand over- and under-coverage obtained in the first phase, in which the departments were considered separately. The outcomes are then used to specify inter-department demand curves for each pair of departments, specifying for each period how many employees of the first department must be serving in the second department.

More specifically, for each day $J_r, r = \{1, \ldots, 7\}$, the optimization problem considered here is constructed as follows. The set $\mathcal{S}_r^{\text{day}}$ of anonymous shifts is given by $\mathcal{S}_r^{\text{day}} = \{s \in \bigcup_{d \in D}(\mathcal{S}_d^{\text{dep}} \cup \mathcal{S}_d^{\text{ex/tr}}) : \text{DAY}(\text{STA}(s)) = J_r\}$, which are all shifts starting at day J_r. Introduce a non-

negative integer variable x_s for each shift $s \in \mathcal{S}_r^{\text{day}}$ indicating how many employees work shift s and two non-negative variables y_{pd}^- and y_{pd}^+ for each period $p \in P$ with $\text{DAY}(p) = J_r$ capturing the under- and over-coverage of the demand. Then solve the following MILP for day $J_r, r = \{1, \ldots, 7\}$:

$$\text{Minimize} \sum_{s \in \mathcal{S}_r^{\text{day}}} c_s x_s + \sum_{\substack{p \in P: \\ \text{DAY}(p) = J_r}} \sum_{d \in D} (c_d^{\text{un}} y_{pd}^- + c_d^{\text{ov}} y_{pd}^+) \tag{4.4a}$$

subject to

$$\sum_{\substack{s \in \mathcal{S}_r^{\text{day}}: \\ p \in P(s,d)}} x_s - y_{pd}^+ + y_{pd}^- = b_{pd} \qquad \text{for all } p \in P \text{ with } \text{DAY}(p) = J_r \text{ and } d \in D, \tag{4.4b}$$

$$\sum_{s \in \mathcal{S}_r^{\text{day}}: D(s) = d} x_s \leq |\{e \in E_d : J_r \in J(e)\}| \qquad \text{for all } d \in D, \tag{4.4c}$$

$$x_s \in \mathbb{Z}_{\geq 0} \qquad \text{for all } s \in \mathcal{S}_r^{\text{day}}, \tag{4.4d}$$

$$y_{pd}^-,\ y_{pd}^+ \geq 0 \qquad \text{for all } p \in P \text{ and } d \in D. \tag{4.4e}$$

The objective (4.4a) captures the shift costs and the demand under- and over-coverage costs. Constraints (4.4b) link the variables y_{pd}^- and y_{pd}^+ to the variables x_s. For each department d, a constraint in (4.4c) limits the total number of shifts covered by employees of d to the number of employees available at day J_r in d. Finally, constraints (4.4d) and (4.4e) specify the domains of the decision variables.

Note that the partitioning into daily problems introduces some imprecision. Shifts may, for example, start at one day and finish at the next. In the above model, such shifts are only attached to the day at which they start. Hence, a potential coverage for periods of the next day are not captured by the model. The partitioning is, however, needed to get manageable MILPs.

Figure 4.4 depicts the solutions obtained by model (4.4) for our example. We observe that the demands of both departments are perfectly covered. Among the 29 selected shifts, there are five transfer and one external shifts.

Given the solutions of model (4.4) for all days $J_r, r = \{1, \ldots, 7\}$, let $\mathcal{S}^{\text{sel}}$ be the set of all selected transfer and external shifts. We derive inter-department demands from set $\mathcal{S}^{\text{sel}}$ as

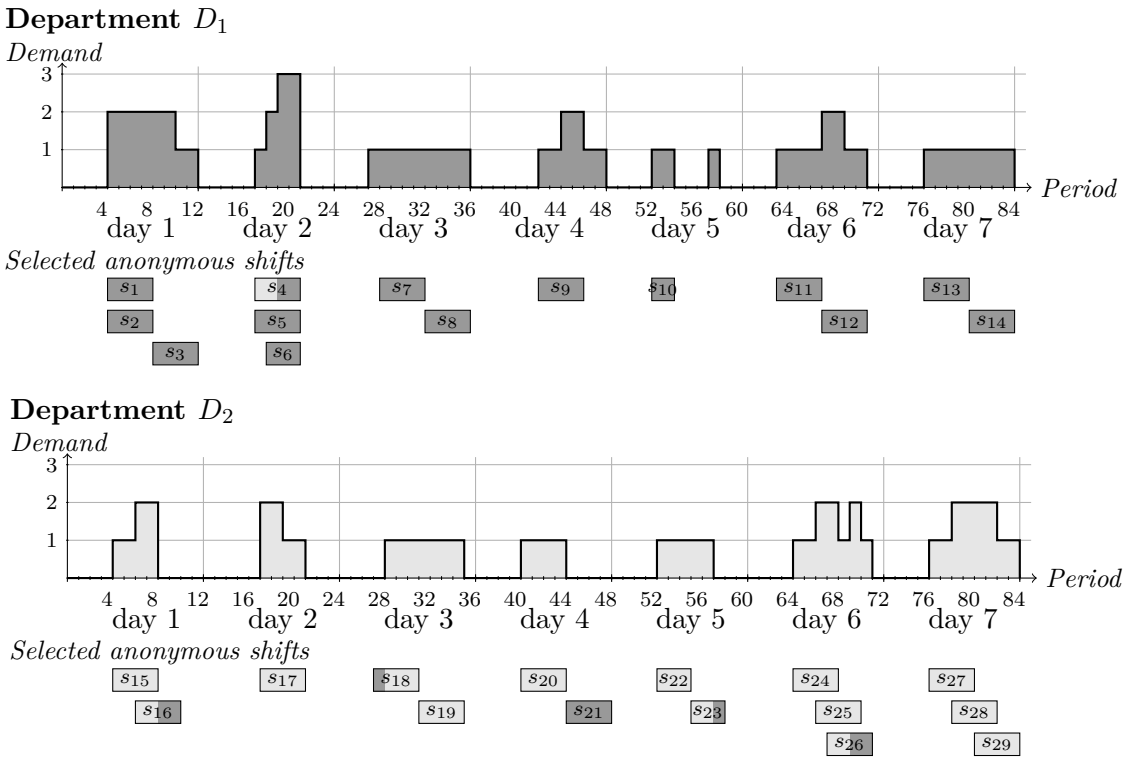

Figure 4.4 An optimal solution of model (4.4) for each day of our example. The external and transfer shifts are depicted in the department providing the employees. The colors of the shifts indicate where the employee is working: dark gray for department D_1, light gray for D_2.

follows. Each department $d \in D$ is assigned to cover a demand of

$$b_{pdd'} = |\{s \in \mathcal{S}^{\text{sel}}, D(s) = d \text{ and } p \in P(s, d')\}| \tag{4.5}$$

for any other department $d'(\neq d)$ in period p, i.e., it is required to transfer $b_{pdd'}$ of its employees to department d' in period p. The right-hand side of equation (4.5) counts how many shifts with department of origin d are selected to cover some demand of department d' in period p. Finally, the intra-department demand b_{pdd}, i.e., the demand of department d that must be covered by its internal employees, is given by

$$b_{pdd} = b_{pd} - \sum_{d' \in D \setminus \{d\}} b_{pd'd}, \tag{4.6}$$

which is the total demand minus the demand transferred to the other departments via the inter-department demands $b_{pd'd}, d' \in D \setminus \{d\}$.

Consider the solution of our example given in Figure 4.4. A demand of one employee is transferred from department D_1 to D_2 in the periods 9, 10, 28, 45 to 48, 58, 70, and 71, and vice versa, a demand of one employee is transferred from D_2 to D_1 in the periods 18 and 19.

4.4.3 Third phase: Department-per-department optimization

In the third and final phase of MP-DH, we decompose the initial employee scheduling problem into (personalized) mono-department scheduling problems where the possibility to transfer employees between departments is reflected by the inter-department demands derived in the previous phase.

Specifically, for each department $d \in D$, the following optimization problem is addressed. First, generate all feasible (personalized) internal shifts that cover at least one period of the internal demand b_{pdd}. Then, create all feasible external and transfer shifts that cover at least one period of the external demand $b_{pdd'}$ in a different department d'. Let $\mathcal{S}_d^{\text{pers}}$ be the so-obtained set of shifts, and define $\mathcal{S}_{de}^{\text{pers}} \subseteq \mathcal{S}_d^{\text{pers}}$ to be the subset of shifts of employee $e \in E$. For each shift $s \in \mathcal{S}_d^{\text{pers}}$, introduce a binary variable x_s taking value 1 if s is selected and 0, otherwise. To capture the over- and under-coverage of the inter- and intra-department demands, introduce two non-negative variables $y^-_{pdd'}$ and $y^+_{pdd'}$ for each period $p \in P$ and

department $d' \in D$. Finally, solve the following MILP for department $d \in D$:

$$\text{Minimize} \sum_{s \in \mathcal{S}_d^{\text{pers}}} c_s x_s + \sum_{p \in P} \sum_{d' \in D} (c_d^{\text{un}} y_{pdd'}^- + c_d^{\text{ov}} y_{pdd'}^+) \tag{4.7a}$$

subject to

$$\sum_{\substack{s \in \mathcal{S}_d^{\text{pers}}: \\ p \in P(s,d')}} x_s - y_{pdd'}^+ + y_{pdd'}^- = b_{pdd'} \qquad \text{for all } p \in P \text{ and } d' \in D, \tag{4.7b}$$

$$\sum_{\substack{s \in \mathcal{S}_{de}^{\text{pers}}: \\ \text{DAY}(\text{STA}(s))=j}} x_s \leq 1 \qquad \text{for all } e \in E_d \text{ and } j \in J(e), \tag{4.7c}$$

$$\sum_{s \in \mathcal{S}_{de}^{\text{pers}}} |P(s)| x_s \leq t_e^{\max} \qquad \text{for all } e \in E_d, \tag{4.7d}$$

$$\sum_{\substack{s \in \mathcal{S}_{de}^{\text{pers}}: \\ \{p_k, \ldots, p_{k+r^{\min}}\} \cap P(s) \neq \emptyset}} x_s \leq 1 \qquad \text{for all } e \in E_d \text{ and } k \in \{1, \ldots, |P| - r^{\min}\}, \tag{4.7e}$$

$$x_s \in \{0,1\} \qquad \text{for all } s \in \mathcal{S}_d^{\text{pers}}, \tag{4.7f}$$

$$y_{pdd'}^-,\ y_{pdd'}^+ \geq 0 \qquad \text{for all } p \in P \text{ and } d' \in D. \tag{4.7g}$$

Model (4.7) is structurally similar to (4.2) and we therefore only point to the differences. Constraints (4.7b) model the demand balance constraint not only for the internal demand of d but also for the demand that must be fulfilled by employees of d for any other department d'. Further note that constraints (4.7c) to (4.7e) must only be added for the employees belonging to department d.

Given a feasible solution of (4.7) for each department $d \in D$, a feasible solution of the entire problem (4.2) can easily be derived. Indeed, the shifts selected in models (4.7) are all feasible, hence they also belong to set $\mathcal{S}$. Let S be the union of the shifts selected in the provided solutions. Then, the corresponding variables $x_s, s \in S$, take value 1 in (4.2). For each period $p \in P$ and department $d \in D$, the value of the under- and over-coverage variables can be set according to their meaning so that (4.2b) is fulfilled. As the provided solutions satisfy constraints (4.7c) to (4.7e), the corresponding constraints (4.2c) to (4.2e) are fulfilled, too. As a result, the so-obtained schedule S of ESP-IDT is feasible. Note that the cost of schedule S is at most the sum of the costs of the feasible solutions of (4.7). Indeed, it can happen that some over-coverage and under-coverage of demands transferred to different departments cancel out in the overall cost calculations.

In our example, we get the final solution depicted in Figure 4.5 by solving model (4.7) for both

departments. We observe that the demand is perfectly covered in both departments and the inter-department transfer feature is used with one external and five transfer shifts. The total cost of this schedule is 35.7. It is an optimal solution for this instance as proven by solving model (4.2). Note that without the inter-department transfer feature, the optimal value is 247.1 with 22 employee-hours of under-coverage and 2 employee-hours of over-coverage.

4.5 Computational experiments

Extensive numerical tests were executed to assess the validity of MP-DH. This section describes the experimental setting and discusses the obtained results.

4.5.1 Experimental setting

We use and slightly tailor the benchmark instances introduced in Dahmen et al. [22] for our numerical experiments. For completeness, we briefly describe the structure of the instances and refer to their article for further details. In all instances, a time period consists of 15 minutes. The unit penalty cost (per period and employee) for under- and over-coverage is 2.35 and 1.175, respectively, the unit work time cost is 0.0375, and the unit transfer cost is 0.025. The minimum duration of a work block is four time periods, the maximum work time per week and employee is 160 periods, i.e., 40 h, and we set the minimum number of rest periods to 48 periods, which is slightly shorter than the 56 periods applied by Dahmen et al. [22]. On average, an employee is qualified to work for about 38% of all departments.

The instances can be grouped according to their size. The small instances (group 1 of Dahmen et al. [22]) involve 20 employees and up to 5 departments. The shift profiles specify three pre-defined time periods each day at which a shift can start (i.e., at 2 a.m., 10 a.m., and 6 p.m.) and restrict the length of a shift to three alternatives (i.e., 7 h, 8 h, and 9 h). While the restrictive shift profiles do not resemble actual practice in retail stores, these instances are useful for a comparison with proven optimal solutions. The medium-sized instances (groups 2 and 3 of Dahmen et al. [22]) contain between 50 and 400 employees and 5 to 10 departments. We specify that the shift length must be either 7 h, 7 h 30 min, 8 h, 8 h 30 min, or 9 h, and valid shift start times are all full hours between 12 a.m. and 8 p.m. These definitions are slightly more restrictive than in Dahmen et al. [22]. The large instances (groups 4 and 5 of Dahmen et al. [22]) have up to 1000 employees and 25 departments, and their shift profile constraints are the same as for the medium-sized instances.

For each combination of number of departments and number of employees, four instances are created with the four demand profiles proposed by Dahmen et al. [22]. In profile 1, 2, 3, and

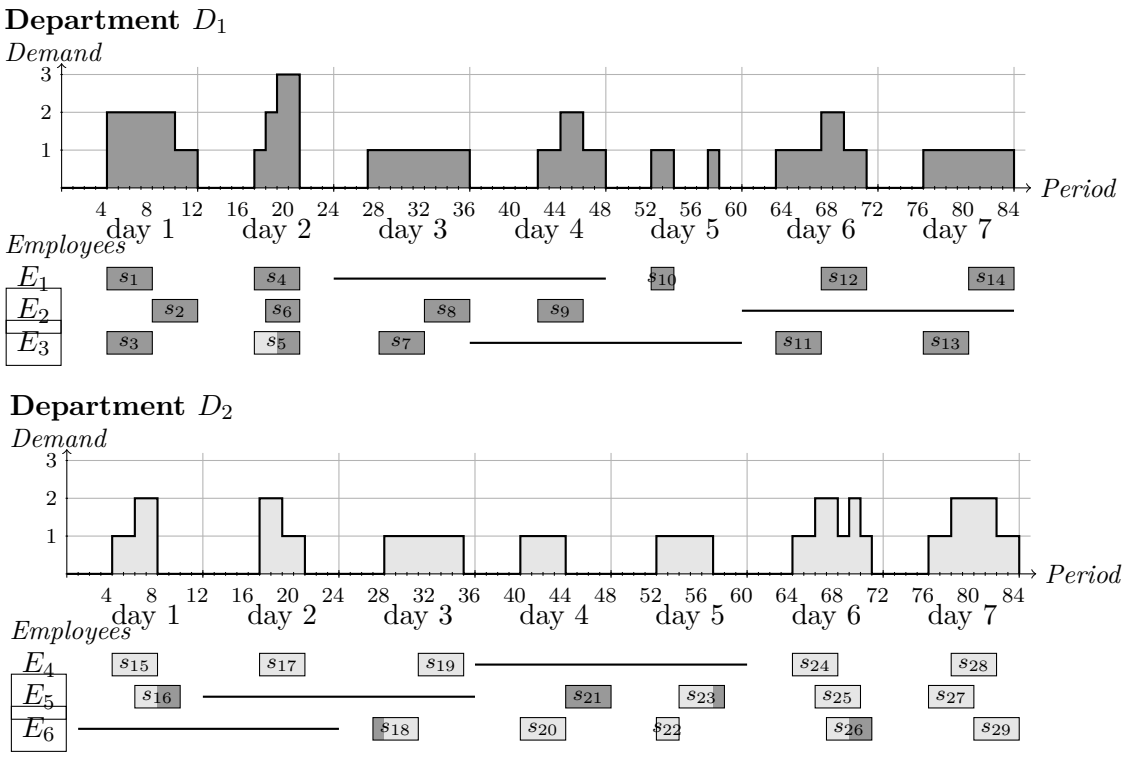

Figure 4.5 The final solution of our example obtained by solving model (4.7) for both departments.

4, the demand can only change approximately every eight, four, two, and one hour(s). Hence, the higher the demand profile number is, the more variability occurs in the demand. Finally, the name of an instance is given by Dx_Ey_Pz, where x is the number of departments, y the number of employees, and z the number of the demand profile.

For each instance, we execute the following four runs. First, we send the (global) model (4.2) without generating external and transfer shifts to the mathematical optimization software XPRESS and solve this version, called global-noTrans for short, with XPRESS' standard branch-and-cut method. This test can be used to assess the value of the inter-department transfer feature. Second, as before, we use model (4.2) and the solver XPRESS, but this time we generate all feasible (internal, transfer, and external) shifts. This run potentially provides a proven optimal solution for the problem under study, and shows how hard it is to directly attack larger instances with model (4.2). Third, we use MP-DH to solve the instance. And fourth, we re-run MP-DH but replace the first phase by simply generating all possible anonymous external and transfer shifts as input for the second phase. We call this version of our method MP-DH-noP1. These tests make it possible to assess the importance and impact of the first phase in MP-DH.

The parameters of MP-DH are set to the same values for all instances. Specifically, β is set to 15 000 000. The MILPs of the second phase are still manageable with this value. Parameter γ is set to 4, which is a reasonable value looking at the given under- and over-coverage costs, and δ is set to 4, which is adequate given the shift profiles.

All tests are executed on a computer with two Intel Xeon 3.50 GHz CPUs and 128 GB RAM. MP-DH is implemented in Java and the branch-and-cut method of XPRESS 8.1.0 solves the mixed-integer linear programs. As in Dahmen et al. [22], we restrict the computation time to two hours for each run. However, when determining the computation time, we exploit the parallelization possibility of MP-DH and assume that the MILPs appearing in all three phases are solved simultaneously. Consequently, the computation time of each phase is determined by the slowest task executed in parallel. Our standpoint is that organizations typically have enough parallel computation power when solving large employee scheduling problems so that what counts at the end is the wall-clock time for computing a schedule. The detailed results are given in Table 4.3 of the appendix, and the following sections provide an in-depth discussion of them.

4.5.2 Computation times and MILP sizes in MP-DH

We first address the computation times of MP-DH. They are mainly determined by the time it takes to solve the corresponding MILP models as the other operations are executed within

a few milliseconds. We therefore analyze the size of the MILP models (in terms of number of variables) and the computation time needed to solve them. Figure 4.6 illustrates these numbers in box plots. The first phase of MP-DH is not presented as the MILPs of this phase are solved to optimality within a second even for the largest instances. This is not surprising as the number of variables is at most about 2000 even for the largest instances, which is a low number for this type of MILP.

Considering the second phase, in which daily, anonymous employee scheduling problems are solved, we observe that it is executed within a few seconds for all small and medium-sized instances. Even the MILPs of the largest instances are solved within two minutes. This is interesting since the number of variables can be quite large for those instances. Indeed, the largest MILP of this phase has about 1.5 million variables. We wish to emphasis that the computation times could be reduced further by stopping the MILP search after some pre-defined time or by reducing the anonymous transfer and external shifts generated in the first phase. When applied carefully, this should only marginally affect the solution quality of MP-DH. For example, one may decide to delete some or all critical time periods derived in the first phase from over-coverage periods if the MILP in the second phase is too large to be solved in a reasonable amount of time. We therefore conclude that the second phase is reasonably simple to solve or can easily be adjusted if the solution process takes too much time.

When looking at the third phase, in which personalized, mono-department scheduling problems are solved, we first observe that the small instances are simple. Indeed, all instances are solved within a second. In good part, this can be explained by the small number of variables, which is at most about 3000. Larger instances, however, need a substantially larger effort to be solved. Indeed, the computation times for medium-sized instances are typically about one hour, and large instances usually take about 1.5 hours of computation time. For these instances, the number of variables is mostly between 10 000 and 30 000. This number can, however, be substantially higher, particularly when large amount of inter-department demands are determined in the second phase. Similarly as in the second phase, the number of variables, and thus the computation time, can be reduced by carefully restricting the set of personalized transfer and external shifts generated in the beginning of the third phase. For example, for each interval with an external demand, one may only create the corresponding personalized shifts for a small subset of the (qualified) employees.

Looking at all three phases of MP-DH together, we conclude that the computation times substantially depend on the size of the instances. They are low for small instances, moderate for medium-sized instances, and quite high for large instances. However, MP-DH can easily

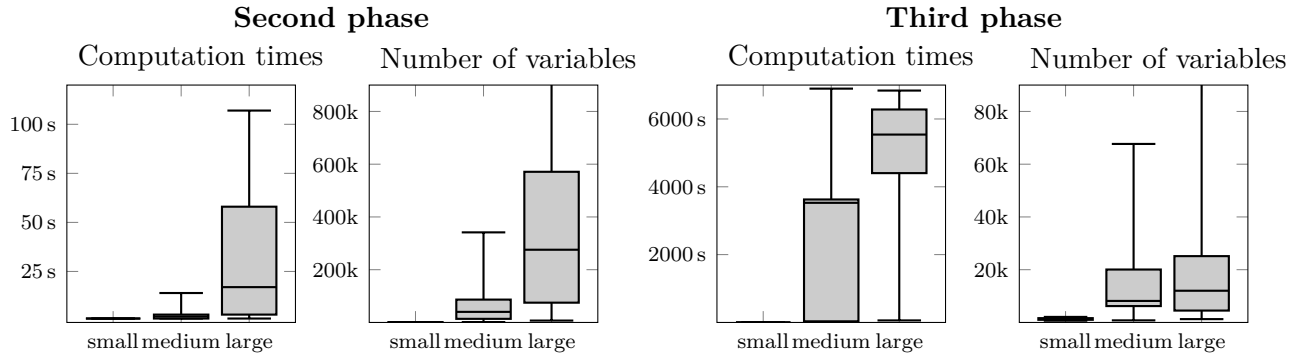

Figure 4.6 Box plots of the computation times and the number of variables of the MILPs solved during the second phase (left) and the third phase (right) of MP-DH. Some box plots are cropped for readability purposes.

be customized and tailored so that solutions are found within reasonable computation times even for large instances. This is an interesting feature of our approach, particularly with respect to an application in practice.

4.5.3 Value of the inter-department transfer feature

When excluding the possibility of inter-department transfers, all benchmark instances can be solved to optimality within two hours of computation time, see columns 2 and 3 of Table 4.3. These results support our view that including the inter-department transfer feature drastically increases the problem complexity. However, the feature also makes it possible to reduce the overall costs substantially. This can be seen in Figure 4.7, which shows a box plot of the absolute difference between the solution values obtained with MP-DH and those of global-noTrans for the small, medium, and large instances.

We first observe that all values of MP-DH are lower than those of global-noTrans, although

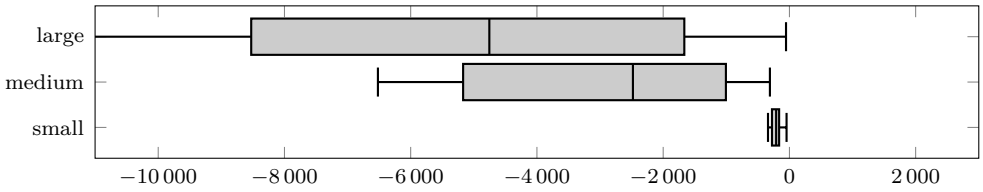

Figure 4.7 Box plots of the absolute differences between the solution values of MP-DH (column 6 in Table 4.3, here *res* for short) and those of model (4.2) without including external and transfer shifts (column 2 in Table 4.3, here *bench* for short). The differences are computed by $res - bench$. Hence, MP-DH is better if this difference is negative.

MP-DH may not be able to find an optimal solution for some instances. Thus, the comparison does not necessarily show the full potential of the transfer feature. Second, as the size of the instances increases, the potential for cost reductions typically increases. But, clearly, not only the size determines the value of the transfer feature. For example, it is obvious that there is not much improvement potential available if one can cover the demand well with internal shifts only, and vice versa, the transfer feature is certainly more valuable if the demands cannot be covered well by the own employees.

To analyze how the inter-department transfers are used, we record the number of internal, external, and transfer shifts present in the solution provided by MP-DH. Figure 4.8 shows the obtained results in box plots. We see that the majority are internal shifts and almost no external shifts are present in the solutions. This can be explained in part by our approach, in which all internal shifts are available in the MILP of the third phase, and by the cost structure. A non-negligible unit transfer cost is charged for each period an employee is working in a non-home department. With these costs, a preference is given to internal shifts while external shifts get quite expensive. The value of the inter-department transfer feature mainly comes from a good use of the transfer shifts, which are used extensively. Indeed, there are up to 20, 700, and 2000 transfer shifts present in the solutions of small, medium and large instances, respectively.

4.5.4 Comparison of MP-DH with proven optimal solutions

An assessment of the solution quality obtained with MP-DH by comparing it with the proven optimum is only possible in the small instances, see columns 4 and 5 of Table 4.3. Indeed, most of the larger instances are actually too big to be read into the MILP solver.

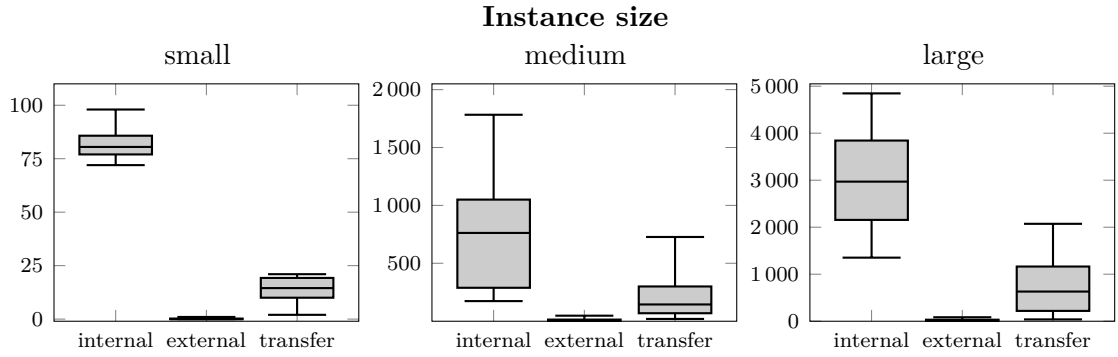

Figure 4.8 Box plots of the number of internal, external, and transfer shifts present in the solutions obtained by MP-DH.

Table 4.1 Relative optimality gaps (in %) of MP-DH for the instances with 20 employees grouped by the number of departments.

Instance	gap	Instance	gap	Instance	gap
D2_E20_P1_s	3.9	D3_E20_P1_s	6.2	D5_E20_P1_s	10.3
D2_E20_P2_s	9.1	D3_E20_P2_s	10.1	D5_E20_P2_s	14.7
D2_E20_P3_s	6.2	D3_E20_P3_s	10.2	D5_E20_P3_s	11.8
D2_E20_P4_s	11.4	D3_E20_P4_s	10.4	D5_E20_P4_s	2.8
average	**7.7**		**9.2**		**9.9**

Table 4.1 displays the relative optimality gaps of MP-DH (in %, computed as $(res - opt)/opt$, where res refers to the value obtained with MP-DH and opt to the optimal value) for the small instances. It can be seen that the optimality gaps are quite large. Indeed, for the instances with 2, 3, and 5 departments, the average gap is 7.7%, 9.2%, and 9.9% respectively. Hence, for small instances, it is certainly preferable to directly use model (4.2) with a state-of-the-art MILP solver. However, when looking at the computation times, we see that it takes a good amount of time (up to 1317 s) for solving these instances to optimality while MP-DH finishes within one second. Hence, if the computation time is critical, one may still prefer to use MP-DH. Furthermore, one may try to increase the quality of the solutions provided by MP-DH with an additional local search that takes the solution of MP-DH for its start. We conducted some preliminary tests with this idea. Using the simple local search proposed by [70], we could decrease the optimality gaps of MP-DH from about 9% to 7%, on average, within few seconds of computation time. In our view, the development of a high-quality local search scheme for the problem under study is an interesting future research project.

We finally emphasis that problems in practice are typically much larger than the small instances of our benchmark set. We introduce them only for the computational tests. Solving practical instances directly with model (4.2) is rarely possible.

4.5.5 Importance of the first phase in MP-DH

To evaluate the impact of the first phase in MP-DH, we compare MP-DH with MP-DH-noP1, in which the first phase is replaced by simply generating all anonymous external and transfer shifts. A remark concerning the computation times is in order. First, we keep a time limit of two hours for solving the MILPs of both remaining phases. We do not subtract the time needed to generate all shifts from this limit. We, however, include these times in the results. Consequently, computation times can be higher than two hours for MP-DH-noP1 in Table 4.3. For one instance, namely D10_E400_P3, no feasible solution is found with

MP-DH-noP1 in two hours. We therefore re-run this instance without computation time limit and report the corresponding results in Table 4.3.

For the comparison of MP-DH with MP-DH-noP1, we compute the absolute difference of the solution values obtained by the two methods and illustrate them in a box plot in Figure 4.9. The following can be observed. Looking at the small instances, no substantial difference between MP-DH and MP-DH-noP1 can be detected. The corresponding computation times, see Table 4.3, are similar, too. When considering the medium-sized instances, we observe that there are larger differences between the results of the two methods. No method is, however, consistently better than the other, and the median value is almost 0. The computation times are also similar, except for instance D10_E400_P3 where MP-DH-noP1 needs about 26 hours to find a first feasible solution in the third phase. This exception is somewhat interesting to analyze further. As in most instances, the computation times in the third phase vary substantially among the departments. Indeed, for all departments except one, the third phase is solved within seven minutes while for the exceptionally difficult department it takes 26 hours to find a first feasible solution. This department is the largest with 91 employees, but it has only two employees more than the second largest department. Also, the number of variables in the MILPs of these two departments are similar (about 55 000 for the largest department and 52 000 for the second largest). Hence, the difficulty cannot be fully explained by the number of employees nor by the size of the MILPs. We further observed that the XPRESS solver consistently uses 26 h for the largest department even with different random seed values. Hence, randomness seems to play a minor role. We suspect that the solver's main difficulties lie in the primal heuristics and in the branching strategies.

For the large instances, MP-DH-noP1 is substantially better than MP-DH. Indeed, it gives lower values in more than 75% of the instances, and in 50% of the instances, the costs are at least lower by 600 than in the solutions obtained by MP-DH. We therefore conclude that MP-DH-noP1 is a valid version of MP-DH, and that the second phase of MP-DH could benefit

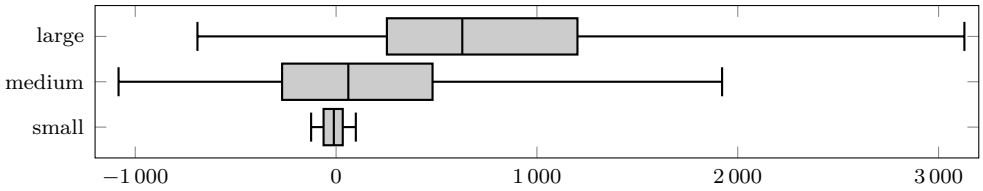

Figure 4.9 Box plots of the absolute differences between the solution values of MP-DH (column 6 in Table 4.3, here *res* for short) and those of MP-DH without the first phase (column 8 in Table 4.3, here *bench* for short). The differences are computed by $res - bench$. Hence, MP-DH is better if the difference is negative.

from a larger set of anonymous transfer and external shifts generated in the first phase.

However, there is also a substantial drawback as seen when comparing the computation times of MP-DH and MP-DH-noP1 for the large instances. They are, on average, about 4800 and 7400 seconds with MP-DH and MP-DH-noP1, respectively. This increase is substantial, and as we see when comparing our results with values from the literature in the next section, MP-DH-noP1 does not scale so well with increasing instance size, showing that it is important to have the first phase in MP-DH when tackling very large instances.

We finally emphasis that a valid option is to combine MP-DH-noP1 with MP-DH as follows. In parallel, we compute the transfer and external shifts with MP-DH and MP-DH-noP1. If the total number of shifts (internal, transfer, and external) of MP-DH-noP1 is below a given threshold –we could use β for this purpose–, we continue with this set in the second phase, and, otherwise, we take the set from MP-DH.

4.5.6 Comparison of MP-DH with literature results

To compare MP-DH with an approach from the literature, we slightly adjust MP-DH so that it exactly addresses the same problem as Dahmen et al. [22]. More specifically, for each employee, we declare that exactly one shift must be executed during a workday, we introduce a constraint restricting the portion of the work time spent in external departments, and we specify department-dependent transfer costs. Introducing these changes in MP-DH is quite straightforward and we therefore omit the details here. Although MP-DH is not tailored to the optimization problem given in Dahmen et al. [22], this comparison helps to assess the quality of MP-DH and shows that our method is quite flexible with respect to the specific problem setting.

We use the medium and large instances of Dahmen et al. [22] for the numerical experiments. We execute one run with MP-DH and MP-DH-noP1 using the same experimental environment as for the other tests. Note that the smallest instances are excluded as they are of little practical relevance and serve in Dahmen et al. [22] mainly for a comparison with proven optimal values.

The detailed results of our experiments are given in Table 4.4 of the Appendix. This table also lists the results of the integrated heuristic (IH) and the time decomposition heuristic (TDH) from Dahmen et al. [22], with which we compare our results.

We first look at the results of MP-DH-noP1. We see that, for the medium-sized instances, they are often better than the results of MP-DH. However, we also observe that MP-DH-noP1 cannot provide a solution for the large instances. There is a simple reason. The number

of transfer and external shifts is simply too large to be handled in the MILP of the second phase. This is due to the quite unrestrictive shift profile rules of Dahmen et al. [22]. The shift profiles are, for example, more restrictive in our main test setting as mentioned in Section 4.5.1. We conclude that the first phase in MP-DH is absolutely necessary for large instances with somewhat unrestrictive shift profiles.

We then look at the results of MP-DH. For comparison purpose, we compute the relative gaps (in %) between the solution values of MP-DH and those of IH and TDH. The obtained values are listed in Table 4.2, grouped by instance size and demand profile type. For example, the relative gaps for the instance D5_70_P2 are given in the line named D5_E70 and columns P2.

We observe that IH and TDH are generally better than MP-DH for medium-sized instances. Particularly in instances with low demand volatility, i.e., with demand profiles 1 and 2, the two methods of Dahmen et al. are at an advantage. The performance difference is quite high. For example, MP-DH provides solutions with 76.8% higher costs than IH averaged over the medium-sized instances with demand profile 1. However, the performance difference between MP-DH and IH/TDH decreases as the demand volatility and the instance size increase. With more than 300 employees and demand profiles 3 and 4, MP-DH matches or outperforms IH and TDH in all instances and is substantially better than IH and TDH in most of these instances. For example, MP-DH decreases the costs by 30.0% on average when compared with TDH for large instances with demand profile 3.

Based on Table 4.4, we determine that the computation times are 2599 s, 5763 s, and 4000 s for MP-DH, IH, and TDH respectively, averaged over the medium-sized instances and 3651 s, 6900 s, 5360 s averaged over the large instances. Also, MP-DH, IH, and TDH are the fastest among the three methods in 37, 0, and 11 instances, respectively. Hence, we conclude that MP-DH has an edge over IH and TDH when considering the computation time.

These results are no surprise. The methods of Dahmen et al. [22] reduce the computational burden by aggregating consecutive periods of the planning horizon. By construction, this aggregation works better if the demand is quite stable, which could be seen in our comparison. Second, their methods do not decompose the overall problem by department unlike MP-DH, which applies this in its first and third phase. Hence, when the number of departments and employees increase, MP-DH can keep the computation times at reasonable levels by benefiting from the decomposition and parallelization possibilities, while the approaches of Dahmen et al. [22] start having difficulties with the computational burden. For example, TDH could not produce a solution for instance D20_800_P1 within a computation time of two hours.

Table 4.2 Relative gaps (in %) between the results of MP-DH and those of IH and TDH, given by $100(res - bench)/bench$, where res and $bench$ refer to the values obtained with MP-DH and IH/TDH, respectively. No solution was given for D20_800_P1 with TDH. We exclude this instance in the average value computations.

	MP-DH versus IH				**MP-DH versus TDH**			
	P1	P2	P3	P4	P1	P2	P3	P4
medium								
D5_E50	64.2	39.6	56.4	7.5	49.4	31.4	46.5	3.0
D5_E70	87.6	44.4	46.7	26.3	81.5	39.9	39.1	19.0
D5_E200	78.5	52.9	25.6	28.7	78.3	45.7	20.8	24.9
D10_E200	70.3	26.2	21.4	19.4	63.4	21.9	18.3	17.0
D10_E300	41.0	40.2	20.6	16.9	38.8	56.1	24.8	15.6
D10_E400	119.0	22.7	-8.2	0.0	113.2	31.9	-8.7	0.9
average	**76.8**	**37.7**	**27.1**	**16.5**	**70.8**	**37.8**	**23.5**	**13.4**
large								
D20_E400	24.8	5.9	-24.4	-8.7	21.1	4.6	-24.7	-8.5
D20_E600	22.6	23.2	-6.6	-18.0	26.3	36.7	-0.8	-9.6
D20_E800	25.8	-31.0	-38.0	-20.9	-	-77.2	-29.8	-12.6
D20_E1000	-10.0	-36.5	-36.9	-27.7	2.8	-27.6	-31.0	-16.8
D25_E800	-64.0	1.5	-23.7	-17.8	-63.9	4.8	-21.5	-10.4
D25_E1000	-55.3	4.4	-72.8	-24.5	-54.5	8.3	-72.0	-19.5
average	**-9.4**	**-5.4**	**-33.7**	**-19.6**	**-13.7**	**-8.4**	**-30.0**	**-12.9**

We conclude that MP-DH can successfully be applied in the setting of Dahmen et al. [22], thus proving its flexibility. Typically, retail stores have quite a large number of employees and a substantial volatility in the demands. Hence, we are convinced that MP-DH is a valid alternative to the methods IH and TDH in practically relevant instances.

4.6 Concluding remarks

MP-DH enables to find good solutions for large ESP-IDT instances within reasonable computation times. The method scales well with the size of instances since it benefits in each of its three phases from problem decompositions and simplifications. MP-DH proved to be valuable especially for large instances with highly variable demands, which are settings occurring frequently in retail stores in practice.

A specific feature of MP-DH is its adaptability. On the one hand, if MP-DH struggles with solving its second phase, one can adjust the anonymous external and transfer shifts generated in the first phase and select a manageable subset of those. For example, one may simply exclude all shifts generated due to an over-coverage of the demand. On the other hand, if MP-DH solves the second phase rapidly, one may increase the number of shifts considered in this phase. In the extreme case, one may simply omit the first phase and generate all feasible external and transfer shifts as second phase input. The third phase, which is the most critical with respect to the computation time, is also adaptive. One can, for example, try to reduce the size of the MILPs of this phase and so the computational burden by omitting a selected subset of external and transfer shifts. This should, however, be carefully applied as it may deteriorate the quality of the solutions.

MP-DH is also flexible with respect to the addressed employee scheduling problem as shown by our computational results with the problem setting of Dahmen et al. [22]. We think that, independent of the specific restrictions and rules, MP-DH performs well for multi-department employee scheduling problems in which internal shifts are favored over inter-department shifts.

In future work, one may try to study and improve some specific steps of MP-DH. First, MP-DH-noP1 has an edge over MP-DH in some larger instances. This points to improvement potential in the generation of the anonymous external and transfer shifts in the first phase of MP-DH. Second, as said before, MP-DH is adaptive. Hence, one may study how to adapt it to the characteristics of specific instances. Another interesting avenue of research consists of developing a local search method for ESP-IDT. As discussed in the computational results, preliminary tests have shown that such an approach can improve the solutions of MP-DH

within seconds. However, further research is needed to establish local search methods that are both effective and efficient for large ESP-IDT instances.

Appendix

The detailed computational results are presented in Tables 4.3 and 4.4. The former gives the results obtained with our main experimental setting and is structured as follows. The instance names are given in the first column. For each instance, the solution value and the total computation time is specified for each of the four runs (global-noTrans, global, MP-DH, and MP-DH-noP1). The instances are grouped and sorted according to their size. Table 4.4 present the results obtained in the setting of Dahmen et al. [22] using a similar structure as in Table 4.3. It shows the solution values and computation times of our runs with MP-DH and MP-DH-noP1 as well as the benchmark values of Dahmen et al. [22] obtained with their integrated heuristic (IH) and the time decomposition heuristic (TDH).

Table 4.3: Detailed results for our main experimental setting. Solution values rounded to one decimal place, computation times given in seconds and rounded to the nearest integer. The best values among MP-DH and MP-DH-noP1 are highlighted in boldface. Average solution values and computation times are given for the groups of small, medium and large instances.

instance	**global-noTrans**		**global**		**MP-DH**		**MP-DH-noP1**	
	value	*time*	*value*	*time*	*value*	*time*	*value*	*time*
small								
D2_E20_P1	1613.8	0	1509.2	4	1568.5	0	**1560.8**	0
D2_E20_P2	2030.7	1	1652.6	146	**1802.7**	1	1829.7	1
D2_E20_P3	1434.5	0	1081.6	156	**1148.6**	1	1224.5	1
D2_E20_P4	1290.0	0	998.5	38	1112.5	0	**1108.0**	0
D3_E20_P1	1757.3	0	1609.7	48	1710.0	0	**1611.7**	0
D3_E20_P2	2137.0	1	1727.1	139	**1901.8**	0	1962.1	1
D3_E20_P3	1838.8	0	1373.5	85	1514.1	0	**1451.2**	0
D3_E20_P4	1765.7	0	1352.0	34	1492.4	0	**1469.1**	0
D5_E20_P1	1870.2	0	1547.1	24	1707.1	0	**1612.5**	1
D5_E20_P2	1933.7	0	1517.9	1317	**1741.1**	0	1809.5	1
D5_E20_P3	1737.3	0	1250.3	471	**1398.0**	0	1522.2	1
D5_E20_P4	1909.6	0	1698.4	191	**1746.5**	0	1792.2	1
average	*1776.6*	*0*	*1443.1*	*221*	*1570.3*	*0*	*1579.5*	*1*

Continued on next page

Table 4.3: Detailed results for our main experimental setting. Solution values rounded to one decimal place, computation times given in seconds and rounded to the nearest integer. The best values among MP-DH and MP-DH-noP1 are highlighted in boldface. Average solution values and computation times are given for the groups of small, medium and large instances. (continued)

instance	global-noTrans		global		MP-DH		MP-DH-noP1	
	value	*time*	*value*	*time*	*value*	*time*	*value*	*time*
medium								
D5_E50_P1	1282.0	7	-	-	**972.9**	3	1395.6	65
D5_E50_P2	2316.5	1	-	-	**1274.1**	9	1389.3	46
D5_E50_P3	2094.0	1	-	-	**1228.1**	17	1440.0	3391
D5_E50_P4	2075.3	2	-	-	**1220.7**	11	1445.6	45
D5_E70_P1	1708.4	7	-	-	**1345.7**	11	1608.7	90
D5_E70_P2	2938.5	7	-	-	2040.8	3188	**1998.1**	79
D5_E70_P3	2790.7	6	-	-	**1698.2**	23	1948.5	80
D5_E70_P4	2276.1	7	-	-	1649.9	955	**1547.8**	473
D5_E200_P1	5002.7	3619	-	-	**3327.1**	3582	4359.2	4650
D5_E200_P2	6350.0	41	-	-	**3682.5**	3582	3969.1	4068
D5_E200_P3	6616.5	156	-	-	**3580.7**	3618	4663.7	3916
D5_E200_P4	5991.6	126	-	-	4276.4	2837	**4197.2**	3719
D10_E200_P1	5012.1	29	-	-	2645.2	37	**2345.6**	148
D10_E200_P2	6539.1	3497	-	-	3874.8	3558	**3382.8**	193
D10_E200_P3	7081.6	80	-	-	4489.8	3546	**4092.3**	471
D10_E200_P4	6642.8	9	-	-	4589.6	254	**4113.2**	228
D10_E300_P1	8250.3	3508	-	-	5008.5	3550	**4398.9**	3078
D10_E300_P2	11314.2	3439	-	-	4921.6	3671	**3801.7**	3775
D10_E300_P3	11412.1	11	-	-	5334.7	3668	**4591.7**	1935
D10_E300_P4	11028.9	8	-	-	5378.8	3532	**5100.1**	430
D10_E400_P1	11881.3	2895	-	-	6546.9	3919	**4625.0**	3825
D10_E400_P2	10076.0	343	-	-	**4961.7**	6897	5368.6	4130
D10_E400_P3	12517.2	221	-	-	6553.8	6349	**4791.5**	95438
D10_E400_P4	12405.2	816	-	-	**5886.0**	5926	6491.3	3599
average	*6483.4*	*785*			*3603.7*	*2614*	*3461.1*	*5745*
large								
D20_E400_P1	5727.5	5766	-	-	**4359.4**	60	4608.5	7037
D20_E400_P2	4135.7	5139	-	-	4026.1	799	**3593.8**	8030

Continued on next page

Table 4.3: Detailed results for our main experimental setting. Solution values rounded to one decimal place, computation times given in seconds and rounded to the nearest integer. The best values among MP-DH and MP-DH-noP1 are highlighted in boldface. Average solution values and computation times are given for the groups of small, medium and large instances. (continued)

instance	global-noTrans		global		MP-DH		MP-DH-noP1	
	value	*time*	*value*	*time*	*value*	*time*	*value*	*time*
D20_E400_P3	7366.1	69	-	-	5502.3	5837	**5184.4**	7101
D20_E400_P4	6968.0	5368	-	-	5788.6	5160	**4931.5**	6114
D20_E600_P1	12719.7	4857	-	-	7654.6	5372	**7208.3**	7615
D20_E600_P2	14988.5	170	-	-	9514.1	4575	**7519.7**	3801
D20_E600_P3	16312.7	6409	-	-	8304.7	5757	**7627.5**	7209
D20_E600_P4	15284.7	5120	-	-	9888.4	736	**7985.5**	3235
D20_E800_P1	18023.5	1784	-	-	10448.3	6358	**9332.6**	8014
D20_E800_P2	22349.7	1072	-	-	9698.7	6727	**8583.3**	8640
D20_E800_P3	22707.1	5433	-	-	10693.8	4834	**10116.2**	6165
D20_E800_P4	21105.7	5054	-	-	11023.0	184	**9740.1**	3201
D20_E1000_P1	18299.2	5685	-	-	**10509.9**	5906	11200.2	7815
D20_E1000_P2	28380.6	1386	-	-	11461.1	6851	**10946.0**	7750
D20_E1000_P3	26565.5	4325	-	-	**12505.0**	6319	12819.9	6595
D20_E1000_P4	22859.5	5683	-	-	**12154.3**	5023	12240.8	7559
D25_E800_P1	9728.2	3268	-	-	**8787.4**	6054	9320.4	8664
D25_E800_P2	8110.7	424	-	-	7930.6	5652	**7351.0**	8172
D25_E800_P3	15089.1	6930	-	-	10644.9	6365	**9470.1**	8485
D25_E800_P4	13132.6	5704	-	-	11367.8	5511	**9674.8**	9517
D25_E1000_P1	15718.3	6506	-	-	12108.8	6470	**12050.5**	9174
D25_E1000_P2	10109.5	4287	-	-	10055.8	6481	**9107.7**	9029
D25_E1000_P3	18291.0	6573	-	-	13951.3	3957	**10822.9**	9225
D25_E1000_P4	16480.4	5996	-	-	13148.0	3543	**11344.0**	9981
average	*15435.5*	*4292*			*9646.9*	*4772*	*8865.8*	*7422*

Table 4.4: Detailed results for the setting of Dahmen et al. [22]. Solution values rounded to one decimal place. Computation times given in seconds and rounded to the nearest integer. The best values among MP-DH, MP-DH-noP1, IH, and TDH are highlighted in boldface. Average solution values and computation times are given for the groups of medium and large instances.

Instance	MP-DH		MP-DH-noP1		IH		TDH	
	value	*time*	*value*	*time*	*value*	*time*	*value*	*time*
medium								
D5_E50_P1	951.2	74	924.4	173	**579.3**	3947	636.6	3627
D5_E50_P2	1345.3	46	1121.5	143	**963.9**	618	1023.7	612
D5_E50_P3	1453.3	90	1392.6	219	**929.0**	3689	992.0	3625
D5_E50_P4	1137.5	43	1231.2	159	**1058.3**	1246	1104.2	1217
D5_E70_P1	1235.4	96	1018.8	236	**658.5**	3851	680.6	3758
D5_E70_P2	1980.8	122	1617.8	307	**1371.4**	3649	1416.4	3628
D5_E70_P3	1821.0	152	1832.0	561	**1241.6**	3824	1309.2	3638
D5_E70_P4	1393.2	247	1295.7	401	**1103.1**	3891	1171.1	3657
D5_E200_P1	2851.5	6320	1999.0	4154	**1597.7**	7202	1599.0	4497
D5_E200_P2	3727.7	5761	2893.2	1459	**2438.5**	7202	2558.0	4055
D5_E200_P3	3603.0	4832	3154.0	1659	**2868.1**	7085	2982.1	4233
D5_E200_P4	4052.1	6615	3867.5	9443	**3148.9**	7169	3243.0	3948
D10_E200_P1	2307.9	421	1716.7	2199	**1355.1**	7204	1412.5	3929
D10_E200_P2	3575.5	1001	3377.8	2095	**2834.0**	7205	2934.3	4523
D10_E200_P3	4088.0	644	3935.9	2687	**3368.0**	7206	3455.0	3994
D10_E200_P4	4368.5	638	3940.2	2081	**3659.5**	7206	3732.7	3965
D10_E300_P1	4419.0	4043	4070.7	3139	**3134.0**	6747	3183.0	4448
D10_E300_P2	4312.1	4947	3583.2	3481	3074.8	7210	**2762.0**	5531
D10_E300_P3	5199.5	4338	5133.7	4687	4310.4	7211	**4165.0**	4962
D10_E300_P4	5517.6	888	5500.2	3497	**4720.4**	6110	4771.1	4174
D10_E400_P1	6047.0	5453	3618.4	4613	**2761.0**	7210	2836.4	4633
D10_E400_P2	4755.3	4903	4605.3	5615	3874.9	7211	**3605.8**	5633
D10_E400_P3	5494.1	6891	**5204.2**	9223	5987.5	7212	6016.8	5681
D10_E400_P4	5900.8	3808	**5689.0**	12199	5900.5	7212	5846.3	5278
average	*4665.4*	*3165*	*4198.0*	*4626*	*3748.3*	*7079*	*3726.7*	*4729*
large								
D20_E400_P1	3174.2	1247	-	-	**2543.7**	4948	2621.6	1670
D20_E400_P2	2963.3	1151	-	-	**2799.2**	5369	2833.1	4167

Continued on next page

Table 4.4: Detailed results for the setting of Dahmen et al. [22]. Solution values rounded to one decimal place. Computation times given in seconds and rounded to the nearest integer. The best values among MP-DH, MP-DH-noP1, IH, and TDH are highlighted in boldface. Average solution values and computation times are given for the groups of medium and large instances. (continued)

Instance	MP-DH		MP-DH-noP1		IH		TDH	
	value	*time*	*value*	*time*	*value*	*time*	*value*	*time*
D20_E400_P3	**4667.3**	1241	-	-	6171.4	6100	6197.3	4592
D20_E400_P4	**4946.0**	871	-	-	5417.2	5491	5404.7	4194
D20_E600_P1	5764.6	2346	-	-	4701.0	7204	**4563.4**	6699
D20_E600_P2	9028.9	4288	-	-	7326.9	7204	**6602.9**	5424
D20_E600_P3	**9042.8**	6279	-	-	9685.2	7205	9118.0	6039
D20_E600_P4	**8694.9**	1522	-	-	10608.6	7204	9622.4	5445
D20_E800_P1	9081.5	3258	-	-	**7220.1**	7205	-	-
D20_E800_P2	**8153.6**	4419	-	-	11810.8	7205	35688.3	6830
D20_E800_P3	**10256.9**	5820	-	-	16551.0	7206	14613.7	5851
D20_E800_P4	**10727.9**	6006	-	-	13557.5	7207	12279.5	5833
D20_E1000_P1	8346.4	4568	-	-	9271.9	7206	**8120.6**	5712
D20_E1000_P2	**9319.0**	2783	-	-	14670.1	7206	12880.4	6086
D20_E1000_P3	**12557.3**	4218	-	-	19900.5	7206	18208.9	5751
D20_E1000_P4	**12672.3**	6420	-	-	17533.7	7207	15231.7	6055
D25_E800_P1	**6582.2**	2447	-	-	18300.5	6999	18243.0	4091
D25_E800_P2	5870.1	5607	-	-	5785.4	7208	**5603.2**	5429
D25_E800_P3	**9362.2**	2096	-	-	12270.5	7208	11926.4	6032
D25_E800_P4	**9293.1**	2038	-	-	11308.5	7204	10374.4	5862
D25_E1000_P1	**11015.3**	4169	-	-	24659.0	6999	24232.9	5660
D25_E1000_P2	7510.6	4598	-	-	7193.4	7208	**6935.5**	4657
D25_E1000_P3	**11048.4**	6567	-	-	40578.3	7208	39499.6	4727
D25_E1000_P4	**11888.0**	3659	-	-	15738.0	7204	14758.6	6475
average	*9622.1*	*4097*			*16434.2*	*7172*	*15501.3*	*5545*

CHAPTER 5 A HYBRID HEURISTIC FOR THE EMPLOYEE SCHEDULING PROBLEM WITH DERIVED INTER-DEPARTMENT TRANSFERS

The MP-DH heuristic presented in the previous chapter starts by pre-processing the ESP-IDT data in the first two phases, resulting in an updated set of inter-department and intra-department requirements for each of the problem departments. The third phase solves for each department an employee scheduling problem with derived inter-department transfers (ESP-DIDT), where only transfers for the derived inter-department requirements are considered. The inter-department requirements creation gives the possibility to decompose the multi-department problem into several smaller mono-department problems maintaining the transfer feature of the global problem. This decomposition reduced the size of the optimized problems, transforming the intractable ESP-IDT into several manageable sub-problems. After the decomposition, the single ESP-DIDT remains a large problem, which optimization may need more than one hour to reach optimality for large instances (Table 4.3).

In this chapter, we present an iterative decomposition-based heuristic accelerating the ESP-DIDT computation time, namely, the *Hybrid heuristic (HH)*. *HH* solves, whenever it is possible, the ESP-DIDT without decomposition, i.e., using Model (4.7) which we will refer to as the *basic model*. Only if the basic model execution exceeds a given time limit, the problem employee set is decomposed and schedules for subsets of employees are optimized iteratively using a semi-anonymous employee scheduling problem with derived inter-department transfers (SA-ESP-DIDT).

5.1 Problem statement

We refer to Section 4.4.3 for the formal presentation of the ESP-DIDT as well as the mathematical model. The decomposition heuristic presented in this chapter splits the department employee set into several subsets of size $numEmp$, where $numEmp$ is a user-defined parameter. The heuristic optimizes every subset employee schedules iteratively using the semi-anonymous employee scheduling problem with derived inter-department transfers (SA-ESP-DIDT). Next we formally present the SA-ESP-DIDT model.

5.2 The SA-ESP-DIDT mixed-integer programming formulation

For each department $d \in D$, we define two sets of employees: E_d^{pers} and E_d^{ano}. E_d^{pers} is a set of size $numEmp$ or less depending on the number of the department employees remaining to schedule. All feasible personalized shifts for the employees in E_d^{pers} are enumerated within the SA-ESP-DIDT MILP forming the set $\mathcal{S}_d^{\text{pers}}$. Feasible personalized shifts include all internal shifts covering at least one period of intra-department requirement b_{pdd} (see Equation 4.6), and all external and transfer shifts covering at least one period of the inter-department requirement $b_{pdd'}$ for all departments $d' \in D, d' \neq d$ (see Equation 4.5). Let $\mathcal{S}_{de}^{\text{pers}}$ be a set containing all feasible personalized shifts for an employee $e \in E_d^{pers}$, where $\mathcal{S}_{de}^{\text{pers}} \subseteq \mathcal{S}_d^{\text{pers}}$.

E_d^{ano} contains the employees for whom the schedules are not optimized in the current MILP. All feasible anonymous shifts, aggregating all possible shifts for all employees in E_d^{ano}, are enumerated forming the set $\mathcal{S}_d^{\text{ano}}$. Feasible anonymous shifts include all internal anonymous shifts covering at least one period of the intra-department requirements b_{pdd}, and all external and transfer anonymous shifts covering at least one period of the inter-department requirements $b_{pdd'}$. The set $E_{dj}^{ano} \subseteq E_d^{ano}$ contains the remaining employees who can work during day j.

A semi-anonymous schedule $\mathcal{S}$ consists of a set of assigned personalized shifts for employees in E_d^{pers} and a list of chosen anonymous shifts, where a single anonymous shift can be chosen multiple times. The semi-anonymous schedule cost $c(\mathcal{S})$ equals the sum of the schedule shift costs and any under-coverage or over-coverage penalties. c_d^{un} and c_d^{ov} are the unit penalty cost for one under-covered and over-covered period unit, respectively, for the department d requirement. Let the number of periods in a shift s be n_{sp} with n_{spt} transferred periods. Shift s cost is $c_s = c^{\text{wt}} * n_{sp} + c^{\text{tr}} * n_{spt}$, where c^{wt} is the unit working cost per period, and c^{tr} is the transfer penalty cost per transferred period.

For each shift $s \in \mathcal{S}_d^{\text{pers}}$, we enumerate a binary variable x_s which takes value 1 if the shift s is considered, 0 otherwise. For each shift $s \in \mathcal{S}_d^{\text{ano}}$, let z_s be a non negative integer variable which indicates the number of times shift s is used. We use the two non-negative integer variables $y_{pdd'}^{-}$ and $y_{pdd'}^{+}$ for each period $p \in P$ and department $d' \in D$ to capture the under- and over-coverage of the inter- and intra-department demands.

Using the above notation and that of Chapter 4, we formally introduce the SA-ESP-DIDT

MILP for department d as follows.

$$\text{Minimize} \sum_{s \in \mathcal{S}_d^{\text{pers}}} c_s x_s + \sum_{s \in \mathcal{S}_d^{\text{ano}}} c_s z_s + \sum_{p \in P} \sum_{d' \in D} (c_d^{\text{un}} y^-_{pdd'} + c_d^{\text{ov}} y^+_{pdd'}) \tag{5.1a}$$

subject to

$$\sum_{\substack{s \in \mathcal{S}_d^{\text{pers}}: \\ p \in P(s,d')}} x_s + \sum_{\substack{s \in \mathcal{S}_d^{\text{ano}}: \\ p \in P(s,d')}} z_s - y^+_{pdd'} + y^-_{pdd'} = b_{pdd'} \qquad \text{for all } p \in P \text{ and } d' \in D, \tag{5.1b}$$

$$\sum_{\substack{s \in \mathcal{S}_{de}^{\text{pers}}: \\ \text{DAY}(\text{STA}(s))=j}} x_s \leq 1 \qquad \text{for all } e \in E_d^{pers} \text{ and } j \in J(e), \tag{5.1c}$$

$$\sum_{\substack{s \in \mathcal{S}_d^{\text{ano}}: \\ \text{DAY}(\text{STA}(s))=j}} z_s \leq |E_{dj}^{ano}| \qquad \text{for all } j \in J, \tag{5.1d}$$

$$\sum_{s \in \mathcal{S}_{de}^{\text{pers}}} |P(s)| x_s \leq t_e^{\max} \qquad \text{for all } e \in E_d^{pers}, \tag{5.1e}$$

$$\sum_{s \in \mathcal{S}_d^{\text{ano}}} |P(s)| z_s \leq t_e^{\max} * |E_d^{ano}|, \tag{5.1f}$$

$$\sum_{\substack{s \in \mathcal{S}_{de}^{\text{pers}}: \\ \{p_k,\ldots,p_{k+r^{\min}}\} \cap P(s) \neq \emptyset}} x_s \leq 1 \qquad \text{for all } e \in E_d \text{ and } k \in \{1,\ldots,|P| - r^{\min}\}, \tag{5.1g}$$

$$x_s \in \{0,1\} \qquad \text{for all } s \in \mathcal{S}_d^{\text{pers}}, \tag{5.1h}$$

$$z_s \in \mathbb{Z}_{\geq 0} \qquad \text{for all } s \in \mathcal{S}_d^{\text{ano}}, \tag{5.1i}$$

$$y^-_{pdd'},\ y^+_{pdd'} \geq 0 \qquad \text{for all } p \in P \text{ and } d' \in D. \tag{5.1j}$$

The objective function (5.1a) minimizes the cost resulting from the use of the personalized shifts as well as the anonymous shifts, along with any under and over-coverage. The anonymous shift cost is an approximation of the cost of assigning the corresponding shift to a real employee. Constraints (5.1b) link the under/over-coverage variables $y^-_{pdd'}$ and $y^+_{pdd'}$ to the x_s and z_s shift variables. Constraints (5.1c) limit employees in E_d^{pers} to work a maximum of one shift per day, and constraints (5.1d) limit the sum of the selected anonymous shifts per day to be less than or equal the the cardinality of set E_{dj}^{ano}. Constraints (5.1e) protect the employees in E_d^{pers} from working more than the maximum allowed time $t_e^{\max}$. Similarly, constraint (5.1f) maintains the sum of the durations of all anonymous shifts in a solution to be lower than the $t_e^{\max}$ multiplied by the cardinality of set E_d^{ano}. Constraints (5.1g) ensure a minimum rest duration of $r^{\min}$ for each employee in E_d^{pers} between each pair of successive

shifts. Constraints (5.1h), (5.1i) and (5.1j) specify the domains of the variables.

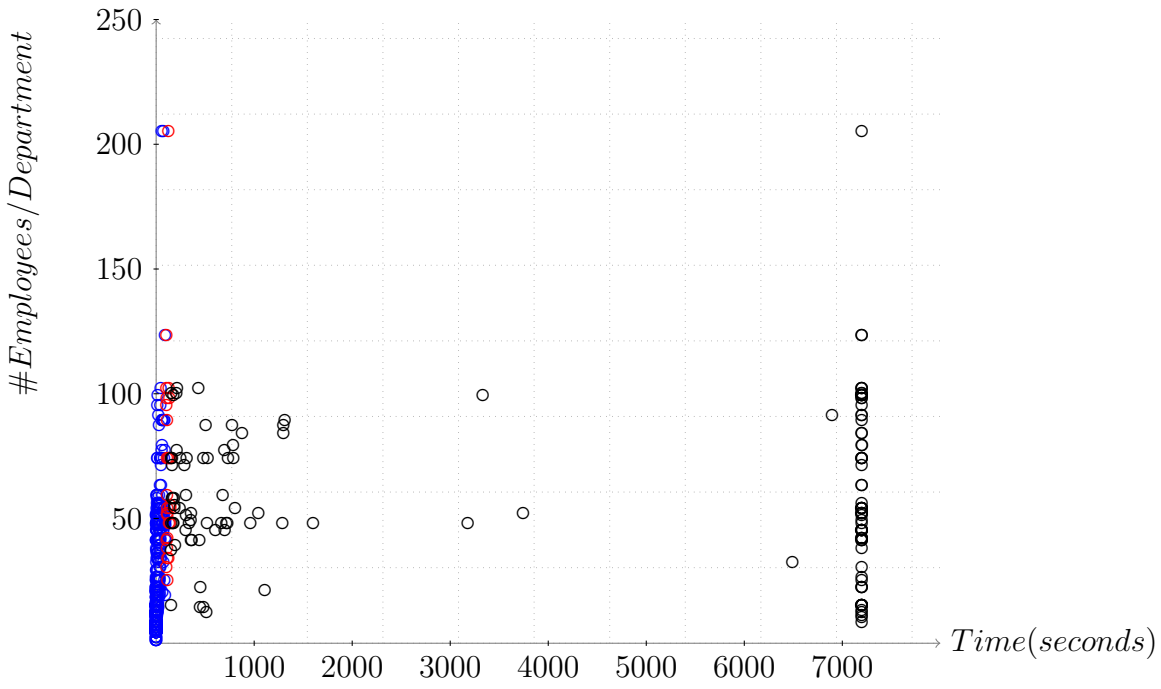

Figure 5.1 Graph showing, for each department within an instance, the basic MILP model computational time vs. the number of employees working for the department.

The development of the semi-anonymous model main purpose is minimizing the size of the *basic model* MILP (4.7) in order to decrease its optimization computation time. This is done by reducing the number of personalized employees, and iteratively running model (5.1) several times. The choice of the number of personalized employees is crucial, the less employees considered at once, the faster the computational times, but the weaker the solution quality. Sensitivity analysis for the number of personalized employees is conducted in Section 5.4.2.

The graph presented in Figure 5.1 shows the computational times achieved by the basic MILP model (4.7) for each department in the medium and large benchmark instances (used for the previous chapter and current chapter computational experiments), with respect to the number of employees per department. A maximum time limit of 2 hours is imposed for the MILP optimization. There are 529 MILPs terminating their computation within 100 seconds (blue circles), 27 MILPs requiring between 100 and 150 seconds (red circles), and 144 MILPs with a computational time exceeding 150 seconds (black circle). From these results, we observe that a high percentage of MILPs involving a large number of employees can be optimized in

short computational time, while departments with a small number of employees can consume hours for being optimized. The graph shows that departments with more than 200 employees are optimized in less than 100 seconds and departments with dozens of employees can reach the time limit of two hours without reaching an optimal solution.

Figure 5.1 suggests that the department ESP-DIDT basic model MILP computational time does not depend only on the number of employees. Thus imposing an employee decomposition for departments with a large number of employees can be unnecessary in most cases.

Also, for an instance with multiple departments, the employees are often unevenly distributed over the departments. During the different department ESP-DIDT parallel optimization, only a small percentage of the departments takes too long and the majority terminates quickly. Thus decomposition is encouraged for only the time consuming departments.

This motivated us to develop a heuristic with a high flexibility on the choice of the $numEmp$ parameter value in the SA-ESP-DIDT model. Indeed, this parameter value controls when to decompose the problem: if $numEmp < |E_d|$, decomposition is activated; otherwise ($numEmp = |E_d|$), no decomposition is used. This gives rise to the hybrid heuristic (HH).

5.3 The hybrid heuristic

The *hybrid heuristic (HH)* uses model (5.1) but changes, on run-time, the parameter $numEmp$ value for the SA-ESP-DIDT optimization. *HH* tries first to solve model (5.1) with all available employees, i.e. $numEmp$ equals the number of available employees and zero anonymous shifts. If the produced MILP computational time exceeds a given time limit t_l, the MILP solution process is canceled, then restarted with fewer personalized employees along with more employees in the anonymous employee set E_d^{ano}.

HH introduces the parameter $empPercent$, which is used to calculate the $numEmp$ value at each iteration. More precisely, $numEmp = empPercent * n_e$, where n_e is the number of available employees.

Algorithm 4 presents *HH*, it takes as a parameter the $empPercent$ value. A new set of employees E_d^{rem} is introduced, which contains all employees whose schedules are not optimized

Algorithm 4: hybridHeuristic(Department d, $EmpPercent$)

```
1  Begin
2      S_d^final ← empty;
3      E_d^rem ← E_d;
4      numEmp ← |E_d^rem|;
5      while E_d^rem not empty do
6          E_d^pers ← choose numEmp employees from E_d^rem;
7          solution ← solve SA-ESP-DIDT for E_d^pers;
8          wait(t_l seconds);
9          gap = current SA-ESP-DIDT MILP optimality gap;
10         if gap < λ% then
11             wait(t_l seconds) or untill the SA-ESP-DIDT MILP is solved, whichever first;
12             solution ←current SA-ESP-DIDT MILP solution;
13             S_semi ← personalized shifts from solution;
14             S_d^final ← S_d^final ∪ S_semi;
15             E_d^rem ← E_d^rem \ E_d^pers;
16             numEmp ← |E_d^rem|;
17             Any covered requirement in R_{d,d'} by S_semi is decremented;
18         else
19             Cancel the SA-ESP-DIDT MILP optimization;
20             numEmp ← empPercent * |E_d^pers|;
21         end
22     end
23     S^final ← ⋃_{d∈D} S_d^final;
24 end
```

yet. At each iteration $E_d^{ano} = E_d^{rem} \setminus E_d^{pers}$. The heuristic starts by solving the SA-ESP-DIDT model (5.1) with the full employee set $E_d^{pers} = E_d^{rem} = E_d$, which is identical to solve the basic model (4.7). The MILP optimality gap is checked after a predetermined computational time limit of t_l seconds: if the gap is less than a given percentage λ (line 10), the MILP continues its computation for another t_l seconds, or until the solver finishes its execution, whichever first. Then, the optimized personalized shifts are preserved in the final schedule shift set S_d^{final} (line 14). The employees for whom the schedule is successfully optimized are removed from the remaining employee set E_d^{rem} (line 15). On line 16, $numEmp$ is set to the cardinality of the full remaining employee set $|E_d^{rem}|$, so the next iteration starts by trying to optimize the whole remaining employee schedules. Finally, the department requirement is updated by removing any requirement covered by a personalized shift (line 17), and a new iteration of *HH* is ready to start.

If the condition at line 10 fails, i.e. the optimality gap is greater than or equal to λ% after

t_l seconds, the *HH* cancels the current MILP execution (line 19). This iteration is called a failed iteration. Then the number of personalized employees is decreased: the current value of *numEmp* is multiplied by *empPercent* resulting in a smaller *numEmp* value (line 20). Consequently, the next iteration optimizes a SA-ESP-DIDT with fewer number of personalized employees. In a failed iteration, the employee set E_d^{rem} is not updated, as no employee schedule is fixed.

When *HH* optimizes department d employee schedules, let D'_d be the set of departments for which department d has inter-department transfer requirements during at least one period $p \in P$:

$$b_{pdd'} > 0 \qquad \forall d' \in D'_d$$

For an employee e with home department $d_e^h = d$, and qualified departments D_e, let D'_e be the set of departments for which employee e is qualified and department d has inter-department transfer requirements, i.e.

$$D'_e = D_e \cap D'_d$$

After a failed iteration, when $numEmp < |E_d^{rem}|$, the choice of the set E_d^{pers} employees, at line 6 of Algorithm 4, is done as follows. The remaining employees in set E_d^{rem} are ordered decreasingly by the cardinality of each employee department set D'_e. Then the first *numEmp* employees are chosen. This order gives priority to the transfer requirements to be covered at the early iterations of *HH*.

When the set E_d^{rem} becomes empty, the department d ESP-DIDT is fully solved and S_d^{final} contains the schedules of the current department employees E_d.

Each iteration of *HH*, regardless whether it is optimizing the whole employee set schedules or only a subset, checks the MILP optimality gap after t_l seconds and decides whether to continue and accept the solution, or decrease the number of employees and re-optimize. Note that, whenever it is possible to solve the model (5.1) with all remaining employees quickly (*quickly* is relative, the choice of t_l value is discussed in Section 5.4.3), *HH* abstains from decreasing the number of personalized employees and losing some of the solution quality.

After employee schedules for all departments have been optimized in parallel, all schedules are gathered and the ESP-IDT cost function is calculated as defined in equation (4.1).

5.4 Computational experiments

All tests were executed on a computer having two Intel Xeon 3.50 GHz CPUs and 128 GB RAM. *HH* was implemented in Java and the XPRESS 8.1.0 MILP solver is used to solve the mixed-integer linear programs. The basic model MILP optimization is forced to stop after two hours or when the optimality gap reaches 0.5%, whichever first. This is not applied for *HH*, as the heuristic is capable to finish quickly.

We benefit from the parallelization nature of the SA-ESP-DIDT between the different problem departments. Thus an instance computational time denotes the computational time for the department with the highest time consumption. We present results for the medium and large instances introduced in Section 4.5.1. Smaller instances are not of interest in this chapter as they do not introduce any computational challenge.

This section is organized as follows. Section 5.4.1 introduces the tests experiment settings. Section 5.4.2 presents results for the standalone *SA* heuristic, which applies model (5.1) with a fixed $numEmp$ parameter value. Results of this section helps emphasizing the benefit of the dynamic assignment value of the $numEmp$ parameter in *HH*. Section 5.4.3 presents a sensitivity analysis for the *HH* parameters. Results are finally discussed in Section 5.4.4.

5.4.1 Experimental setting

The MP-DH parameter values are set as follows: $\beta = 15\,000\,000$ (maximum number of enumerated shifts in the second phase), $\gamma = 4$ (minimum number of consecutive over-covered periods in order to consider it a critical interval) and $\delta = 8$ (number of periods used in the transfer shift enumeration before and after the critical interval). HH is controlled by three parameters: $empPercent$, t_l and λ. A sensitivity analysis is conducted and presented next for the first two parameters. For the optimality gap tolerance λ, a value of 0.5% is chosen in order to preserve the solution quality as good as possible.

For the cost function, we use the same costs and penalties as in the previous chapter. A unit working period cost is 0.0375 and a transfer working period gets a penalty of 0.025. Under-coverage and over-coverage unit costs are 2.35 and 1.175, respectively.

To emphasize the strength of *HH*, we compare the *HH* results with the *basic model* solved with XPRESS and stopping whenever the MILP optimality gap reaches 0.5%. This is done in order to prove that the fast execution of *HH* is not caused by an early termination due to the optimality gap parameter $\lambda = 0.5\%$.

5.4.2 *SA* heuristic

To analyze the impact of using a fixed value of the *numEmp* parameter throughout the solution process, we first ran experiments with the SA heuristic. Their results are reported in Tables 5.1 and 5.2 for *numEmp* equal to 10, 20, 30 and 40. For each dataset, the best results are highlighted in bold.

The *SA* heuristic is a powerful tool for decreasing the ESP-DIDT size, but it has two drawbacks. The first drawback is that despite the decrease in the size of the MILP solved for each iteration, it consumes large computational times. The *SA* heuristic is not capable to guarantee fast computation for the ESP-DIDT even for departments with few employees. The second drawback is that even if a department *basic model* would have been solved quickly (before t_l seconds), the *SA* heuristic imposes the employee decomposition, which implies a deterioration in the solution quality without a meaningful gain in the computation time. For example, instance $D10\_E200\_P1$ computational time for the MP-DH using the *basic model* is 19 seconds (Tables 5.4 and 5.5), while the *SA* heuristic optimizing 40 employees per iteration spent 21 seconds, and the solution cost went from 2630 to 2778. Thus the loss in the solution quality is not justified by a significant gain in the computation time.

It is clear that for large instances the computation time is large and can exceed the two-hours limit with $numEmp = 40$. This happens when several iterations achieve the time limit of two hours. In general, larger values for *numEmp* give better solutions, while smaller values for *numEmp* has better computational time. Thus, we choose $numEmp = 40$ employees per iteration for the *SA* heuristic to compare with *HH* in the next section.

5.4.3 Sensitivity analysis of the *HH* parameters

HH is tested using different employee percentages *empPercent* ($25\%, 50\%$ and 75%) and different computational time limits t_l ($50, 100$ and 150 seconds) for the medium and large

Table 5.1 Cost and computation time in seconds for the semi-anonymous model with different number of employees per iteration (*numEmp*) for the medium datasets

	Number of employees per iteration							
	10		**20**		**30**		**40**	
Instance	*value*	*time*	*value*	*time*	*value*	*time*	*value*	*time*
D5_E50_P1	1084	4844	**973**	**4**	**973**	**4**	**973**	**4**
D5_E50_P2	1382	7	**1274**	**11**	**1274**	**11**	**1274**	**11**
D5_E50_P3	1343	8	**1218**	**44**	**1218**	**44**	**1218**	**44**
D5_E50_P4	1276	8	**1221**	**9**	**1221**	**9**	**1221**	**9**
average	*1271*	*1217*	***1171***	***17***	***1171***	***17***	***1171***	***17***
D5_E70_P1	1572	4866	1383	8	**1317**	**8**	**1317**	**8**
D5_E70_P2	2178	**8**	2080	4912	**2010**	4766	**2010**	4144
D5_E70_P3	1875	4550	1690	**29**	**1607**	34	**1607**	35
D5_E70_P4	1694	5239	1524	**427**	**1466**	637	**1466**	519
average	*1830*	*3666*	*1669*	*1344*	***1600***	*1361*	***1600***	***1176***
D5_E200_P1	4250	151	4159	1319	3884	10457	**3601**	**30**
D5_E200_P2	4307	333	4199	7458	4123	5169	**3974**	**285**
D5_E200_P3	4503	**18**	3939	4656	3943	10932	**3737**	5281
D5_E200_P4	4682	8983	**4468**	12386	4629	6240	4679	**3929**
average	*4436*	***2372***	*4191*	*6455*	*4145*	*8199*	***3998***	*2382*
D10_E200_P1	3467	**6**	2960	9	**2777**	13	2778	21
D10_E200_P2	4459	6763	4138	4963	**3743**	**169**	3809	223
D10_E200_P3	4957	**44**	4402	4941	4343	5137	**4329**	5652
D10_E200_P4	5001	**19**	4504	3804	4392	5809	**4363**	6035
average	*4471*	***1708***	*4001*	*3429*	***3814***	*2782*	*3820*	*2983*
D10_E300_P1	6244	**9**	5878	4507	5547	165	**5240**	4801
D10_E300_P2	6031	4987	5339	5194	**4879**	5161	4909	**4293**
D10_E300_P3	6594	5609	5679	**4785**	**5571**	5578	5695	5548
D10_E300_P4	6172	**23**	5866	4799	5535	6437	**5460**	5659
average	*6260*	***2657***	*5691*	*4821*	*5383*	*4335*	***5326***	*5075*
D10_E400_P1	8314	**22**	8086	5555	8016	74	**7674**	6102
D10_E400_P2	6769	**5142**	6124	5189	**5632**	10415	6186	6961
D10_E400_P3	7377	8824	6682	12637	6656	6435	**6629**	**4438**
D10_E400_P4	7216	**54**	7220	4826	**6497**	10291	7040	6085
average	*7419*	***3510***	*7028*	*7052*	***6700***	*6804*	*6882*	*5897*
average	*4281*	*2522*	*3959*	*3853*	*3802*	*3917*	*3799*	*2922*

Table 5.2 Cost and computation time in seconds for the semi-anonymous model with different number of employees per iteration (*numEmp*) for the large datasets

	Number of employees per iteration							
	10		**20**		**30**		**40**	
Instance	*value*	*time*	*value*	*time*	*value*	*time*	*value*	*time*
D20_E400_P1	6510	**10**	5605	5317	5412	21	**4812**	6239
D20_E400_P2	6260	**7**	4852	64	4833	65	**4343**	65
D20_E400_P3	9011	174	8080	5129	7794	**108**	**7561**	118
D20_E400_P4	7431	5314	6557	5888	6242	5352	**6008**	**5266**
average	*7303*	***1376***	*6273*	*4100*	*6070*	*1386*	***5681***	*2922*
D20_D600_P1	11610	**121**	10759	4894	**9663**	1485	9881	1396
D20_D600_P2	12073	**28**	10710	5158	**10324**	5381	10467	5155
D20_D600_P3	11060	**74**	10020	5641	**9456**	5906	9678	4651
D20_D600_P4	12909	**276**	10959	5704	10554	10230	**10517**	10221
average	*11913*	***125***	*10612*	*5349*	***9999***	*5751*	*10136*	*5356*
D20_E800_P1	14715	**5271**	14436	5451	14006	5809	**13318**	6319
D20_E800_P2	13102	**227**	12359	4876	**11489**	11089	11514	6091
D20_E800_P3	14529	**86**	13329	5853	12463	511	**12160**	6057
D20_E800_P4	15057	**108**	13652	5150	12798	6654	**12530**	5893
average	*14351*	***1423***	*13444*	*5332*	*12689*	*6016*	***12380***	*6090*
D20_E1000_P1	14883	5194	15029	**303**	14262	5222	**14218**	5338
D20_E1000_P2	17679	**60**	15266	5230	**14521**	9939	15215	10175
D20_E1000_P3	16797	**112**	15921	5432	15465	5744	**14692**	6214
D20_E1000_P4	16194	**123**	15103	8416	14848	5644	**14141**	10702
average	*16388*	***1372***	*15329*	*4845*	*14774*	*6637*	***14566***	*8107*
D25_E800_P1	12603	**90**	11707	2843	10989	5344	**10584**	6362
D25_E800_P2	12527	**156**	10169	4375	10601	3618	**9988**	6329
D25_E800_P3	13949	**63**	12821	4700	12344	9929	**11599**	5801
D25_E800_P4	14155	**107**	12825	4785	12785	5667	**12149**	6100
average	*13309*	***104***	*11880*	*4176*	*11680*	*6139*	***11080***	*6148*
D25_E1000_P1	15994	**12**	15641	5537	15168	930	**14503**	4763
D25_E1000_P2	14924	**15**	14132	4885	**12534**	4529	12718	5076
D25_E1000_P3	17790	**4642**	15894	6104	**15314**	6027	16077	10358
D25_E1000_P4	20725	**692**	19605	5444	18740	4902	**18441**	5561
average	*17358*	***1340***	*16318*	*5493*	*15439*	*4097*	***15435***	*6439*
average	*13437*	*957*	*12310*	*4882*	*11775*	*5004*	*11546*	*5844*

datasets. Table 5.3 presents the average cost and computation time in seconds over all datasets for the different parameter values.

We observe that the smaller the employee percentage, the better *HH* performs, i.e., it yields better solutions. The reason behind this behavior is the loss of information coupled with the anonymous shift usage. When anonymous shifts are used as an aggregation to a set of employee shifts (E_d^{ano}), the constraints associated with the employee shift maximum durations and with the maximum number of possible shifts per horizon are aggregated (Constraints (5.1f) and (5.1d)). One constraint is not considered: the minimum rest duration between each pair of an employee shifts, as there exist no way to link a couple of anonymous shifts to a single employee. Consequently, when a large employee percentage is optimized during the first iteration, the minimum rest duration constraint would prevent the remaining small number of employees from being assigned some shifts in the next iteration, resulting in costly under-coverage.

For the simplified example of Figure 5.2, department d has 4 employees and the 2-day horizon is discretized in 2-hour periods. Let the minimum rest duration between a pair of successive shifts be 10 hours. The left graph shows the solution of the first *HH* iteration when $empPercent = 75\%$, optimizing schedules for employees e_1, e_2 and e_3. For the next iteration optimizing the remaining employee (e_4) schedule, the shifts with the same start and end periods as those of the anonymous shifts $ano1$ and $ano2$ cannot be used, as the difference between the end of shift $ano1$ and the beginning of shift $ano2$ is 8 hours. A shift starting after 10 hours of shift $ano1$ end can be assigned to employee e_4 but the solution will always have under-coverage.

The right graph shows the result for the first iteration using $empPercent = 25\%$. During the next iteration, the remaining requirements can be covered by several shift combinations of the remaining employees, namely e_2, e_3 and e_4. One possible shift assignment combination is: employee e_2 assigned to shift $ano3$, employee e_3 assigned to shift $ano4$, employee e_4 assigned to shift $ano5$, employee e_3 assigned to shift $ano6$, employee e_2 assigned to shift $ano7$ and employee e_4 assigned to shift $ano8$. Note that such a direct assignment of these selected anonymous shifts is not performed directly in our algorithm. Instead, a MILP needs to be solved. This assignment just shows that there exists one possible solution without under-coverage.

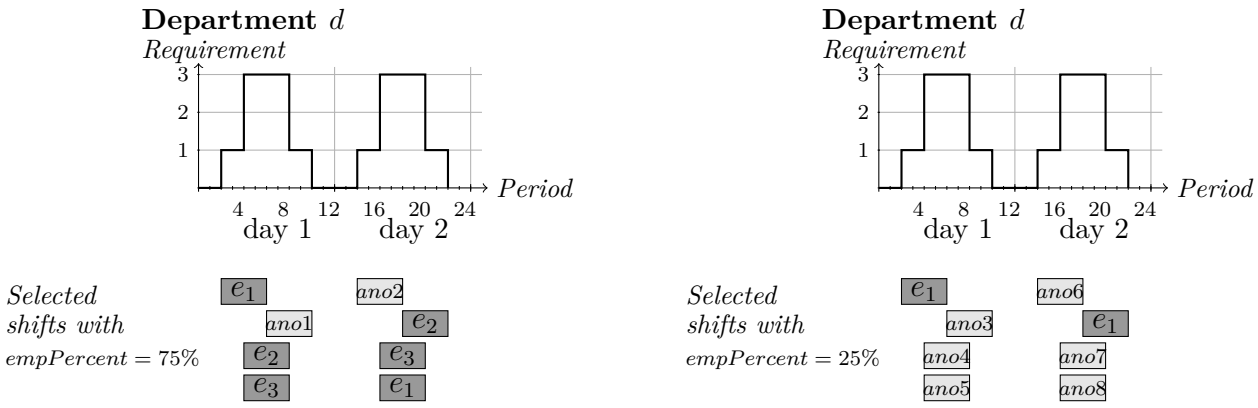

Figure 5.2 Simplified example: *EmpPercent* analysis

Imposing the shortest time limit value $t_l = 50$ seconds does not perform as good as using the 100 or 150 seconds, while it achieves faster computation. There is always a tradeoff between solution quality and computational time. For the results with $empPercent = 25\%$, the average solution cost for $t_l = 100$ seconds is 0.8% higher than the average cost with $t_l = 150$, but it is 34% faster. For the next section, we present the *HH* results using $empPercent = 25\%$ and $t_l = 100$ seconds.

Table 5.3 Average cost and time in seconds for different employee percentages *empPercent*, and time limits t_l for the *HH*.

	empPercent					
	0.25%		**0.5%**		**0.75%**	
t_l (seconds)	*cost*	*time*	*cost*	*time*	*cost*	*time*
50	7017	282	7276	229	7396	270
100	6886	441	7138	345	7229	365
150	6826	592	7011	457	7222	523

5.4.4 Discussion

Tables 5.4 and 5.5 compare the results obtained by the *SA* heuristic with $numEmp = 40$, and *HH*, with $empPercent = 25\%$ and $t_l = 100$, to the basic model results, for the medium and large datasets, respectively. The flexibility in the *numEmp* parameter value choice, as well as monitoring the MILP convergence on run-time by *HH*, outperforms the standalone *SA* heuristic. The best results are highlighted in bold for each dataset.

The average cost achieved by *HH* is 1.9% worse than the basic model for the medium datasets, and 3.1% for the large datasets. With respect to the computational time, *HH* is 86.7% faster than the average basic model computation for the medium datasets and 87.4% for the large datasets.

From Tables 5.4 and 5.5, we see that the *HH* computational time is often very close to a multiple of 100 seconds, as $t_l = 100$. The maximum *HH* computation duration is of 1261 seconds ($\approx$ 21 minutes) for dataset $D10\_E300\_P3$, which means one department had at most 12 SA-ESP-DIDT optimizations. The same dataset needed 6244 seconds ($\approx$ 2 hours) to be optimized with the basic model.

The *HH* shows its success not only with the computation acceleration, but also with maintaining the solution quality within 3% of the basic model solution quality on average.

Table 5.4 Comparing the basic model to the SA and HH heuristics cost and duration for the medium datasets.

	Basic		**SA**		**HH**	
Instance	*value*	*time*	*value*	*time*	*value*	*time*
D5_E50_P1	**973**	4	**973**	4	**973**	**3**
D5_E50_P2	1275	**11**	**1274**	**11**	**1274**	14
D5_E50_P3	1220	**12**	**1218**	44	**1218**	51
D5_E50_P4	1223	11	**1221**	**9**	**1221**	16
average	*1172.75*	***9.5***	***1171.5***	*17*	***1171.5***	*21*
D5_E70_P1	1318	11	**1317**	**8**	**1317**	9
D5_E70_P2	2015	611	**2010**	4144	2068	**256**
D5_E70_P3	1609	**13**	**1607**	35	**1607**	45
D5_E70_P4	1467	**26**	**1466**	519	1467	202
average	*1602.25*	*165.25*	***1600***	*1176.5*	*1614.75*	***128***
D5_E200_P1	**3332**	5042	3601	**30**	3596	359
D5_E200_P2	**3615**	1753	3974	**285**	3798	709
D5_E200_P3	**3425**	4236	3737	5281	3680	**434**
D5_E200_P4	**3922**	4437	4679	3929	4380	**703**
average	***3573.5***	*3867*	*3997.75*	*2381.25*	*3863.5*	***551.25***
D10_E200_P1	**2630**	**19**	2778	21	2642	**19**
D10_E200_P2	**3698**	**129**	3809	223	3702	162
D10_E200_P3	4236	574	4329	5652	**4224**	**245**
D10_E200_P4	**4249**	121	4363	6035	4378	**107**
average	***3703.25***	*210.75*	*3819.75*	*2982.75*	*3736.5*	***133.25***
D10_E300_P1	**4660**	4825	5240	4801	4958	**201**
D10_E300_P2	**4656**	5173	4909	4293	5007	**1028**
D10_E300_P3	**5066**	6244	5695	5548	5297	**1261**
D10_E300_P4	**5186**	4945	5460	5659	5255	**358**
average	***4892***	*5296.75*	*5326*	*5075.25*	*5129.25*	***712***
D10_E400_P1	**6519**	6435	7674	6102	6860	**410**
D10_E400_P2	5546	4547	6186	6961	**5327**	**698**
D10_E400_P3	6554	6349	6629	4438	**5759**	**528**
D10_E400_P4	5868	5763	7040	6085	**5867**	**338**
average	*6121.75*	*5773.5*	*6882.25*	*5896.5*	***5953.25***	***493.5***
average	*3511*	*2554*	*3799*	*2922*	*3578*	*340*

Table 5.5 Comparing the basic model to the SA and HH heuristics cost and duration for the large datasets.

	Basic		**SA**		**HH**	
D20_E400_P1	**4359**	**127**	4812	6239	4839	223
D20_E400_P2	**4028**	547	4343	**65**	4549	310
D20_E400_P3	**7377**	**118**	7561	**118**	**7377**	218
D20_E400_P4	**5777**	**77**	6008	5266	5891	327
average	***5385.25***	***217.25***	*5681*	*2922*	*5664*	*269.5*
D20_D600_P1	**7699**	6617	9881	1396	8551	**327**
D20_D600_P2	**9539**	4782	10467	5155	9635	**664**
D20_D600_P3	**8260**	6156	9678	4651	8468	**482**
D20_D600_P4	**9775**	4757	10517	10221	9891	**492**
average	***8818.25***	*5578*	*10135.75*	*5355.75*	*9136.25*	***491.25***
D20_E800_P1	**10433**	4125	13318	6319	10861	**807**
D20_E800_P2	**9466**	5406	11514	6091	10506	**710**
D20_E800_P3	**10509**	4979	12160	6057	10617	**450**
D20_E800_P4	**11009**	562	12530	5893	11075	**469**
average	***10354.25***	*3768*	*12380.5*	*6090*	*10764.75*	***609***
D20_E1000_P1	**10506**	5957	14218	5338	11662	**315**
D20_E1000_P2	12268	6445	15215	10175	**12102**	**689**
D20_E1000_P3	12705	6235	14692	6214	**12539**	**908**
D20_E1000_P4	**12144**	3883	14141	10702	12311	**578**
average	***11905.75***	*5630*	*14566.5*	*8107.25*	*12153.5*	***622.5***
D25_E800_P1	**8795**	5402	10584	6362	9593	**1000**
D25_E800_P2	**7848**	4466	9988	6329	8517	**602**
D25_E800_P3	**10637**	5598	11599	5801	11363	**633**
D25_E800_P4	**11317**	4846	12149	6100	11531	**318**
average	***9649.25***	*5078*	*11080*	*6148*	*10251*	***638.25***
D25_E1000_P1	**12122**	1669	14503	4763	12175	**218**
D25_E1000_P2	**10491**	6507	12718	5076	10498	**703**
D25_E1000_P3	13705	6626	16077	10358	**13175**	**970**
D25_E1000_P4	**16491**	6192	18441	5561	16947	**600**
average	*13202.25*	*5248.5*	*15434.75*	*6439.5*	***13198.75***	***622.75***
average	*9886*	*4253*	*11546*	*5844*	*10195*	*542*

CHAPTER 6 PARALLEL LARGE NEIGHBORHOOD SEARCH FOR MULTI-JOB EMPLOYEE SCHEDULING PROBLEM

The multi-job employee scheduling problem (MJ-ESP) is slightly different from the multi-department employee scheduling problem tackled in the previous chapters. First, shifts contain a single job, no transfers are allowed. Second, each employee has his own shift profile, i.e., there are no fixed rules for all (shift starting periods and lengths, maximum and minimum working hours per week, ...).

The common aspect between the two problems is that when an instance size increases, its optimization using exact methods becomes impractical. In this chapter we use the large neighborhood search (LNS) metaheuristic to solve large instances of the MJ-ESP. LNS destroys part of a solution, then repairs it again. For the repair procedure we use the WFC commercial MJ-ESP MILP solver developed at Kronos Inc., which makes our heuristic a matheuristic, combining mathematical programming and metaheuristic. Repairing the destroyed part of the solution using mathematical programming is a powerful tool because it results directly with the best possible repaired solution, meanwhile it is not time consuming for small destroyed parts.

The MJ-ESP is presented in Section 6.1, then formally defined in Section 6.2. The LNS heuristic is elaborated in Section 6.3, and its parallel version in Section 6.4. Finally the computational results are presented in Section 6.5.

6.1 The multi-job employee scheduling problem

The MJ-ESP is formally defined by a set of jobs J_g, a set of employees E_g and a set of days D_g containing the days from the beginning to the end of the problem time horizon. The subscript $_g$ denotes *global*, used to distinguish from other sets described later in this chapter.

The problem time horizon is specified by its start day/hour and its end day/hour. The horizon is discretized into 15-minute periods. Let the ordered set P_g contains all the horizon periods. Any period $p \in P_g$ starts and ends at the same day: $day(p) \in D_g$.

A job requirement r_{jp} is the number of employees needed to work on job $j \in J_g$ during the

period $p \in P_g$. Each job j requirement for the whole horizon is presented in the ordered set R_j.

Each employee $e \in E_g$ is qualified for a subset of jobs $J_e \subseteq J_g$. For each job $j \in J_e$, the employee e is qualified to work on this job only during his job availability days $D_{ej} \subseteq D_g$.

Each employee has his own employee profile which specifies the rules governing the employee shifts. An employee profile contains the maximum and minimum hours per week worked by the employee. The maximum working hours is a hard constraint, while the minimum weekly working hours is set as a soft constraint, penalized in the objective function. Minimum and maximum shift duration, working day/hour availability for the whole problem horizon, where days off are set as unavailable days, eligible shift starting periods, and minimum rest duration between successive shifts are also included in each employee profile.

For a given schedule, over-coverage exists during period p for job j if the number of employees assigned to job j during period p is greater than r_{jp}. Over-coverage is allowed but penalized in the objective function. On the other hand, under-coverage is not allowed. In case of unavoidable under-coverage, e.g. employee maximum working hours are already consumed, or no qualified employee for the current job and period exists, *open shifts* are used. Open shifts are anonymous shifts, having their own shift profile as well. Open shifts are used to highlight shifts that can be assigned to additional part-time or on-demand employees.

A shift s, whether assigned to an employee $emp(s) \in E_g$, or anonymous $emp(s) = \phi$, where ϕ represents an anonymous employee, has a starting period $start(s) \in P_g$, an ending period $end(s) \in P_g$, and is serving exactly one job $job(s) \in J_g$. We define a shift by the quadruplet $(emp(s), job(s), start(s), end(s))$. A feasible shift is a shift respecting its associated employee profile. All feasible shifts for each employee $e \in E_g$ are enumerated in a set S_e. Similarly, all possible open shifts are enumerated in the set S_{op}. Let the set of all feasible employee shifts of the problem be $S = \bigcup_{e \in E_g} S_e$. All employee shifts s having $job(s) = j \in J_g$ are grouped in a set $S_j \subset S$. Similarly, all open shifts s assigned to job j are grouped in the set $S_{op,j}$. For each shift $s \in S \cup S_{op}$, let the set $Periods(s) \subset P_g$ be the set containing all time periods covered by the shift s. A shift s belongs to the day of the first period of the shift, i.e. $day(start(s))$; for simplicity, we will use $day(s)$ directly.

A MJ-ESP solution is called a schedule. A schedule $\mathcal{S}$ is the set of personalized shifts and

open shifts that must be executed, $\mathcal{S} \subset S \cup S_{op}$. A schedule is feasible if all employee shifts respect their employee shift profile, and all open shifts respect the open shift profile. The quality of a schedule is evaluated by its cost. Several components form the cost $c(\mathcal{S})$ of a schedule $\mathcal{S}$:

- The shift cost convex function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ adds a non-negative cost $f(\lambda_e)$ to the objective function, for each employee e total working hours λ_e.
- The convex open shift cost function $f_{op} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ adds a non-negative penalty of $f_{op}(\lambda_{op})$ for a total of λ_{op} hours of open shifts in the schedule.
- The convex function $h : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ maps the difference between an employee $e \in E_g$ actual total working hours and his preferred minimum working hours δ_e to the non-negative penalty value $h(\delta_e)$, only in case of positive δ_e values. $h(\delta) = 0, \forall\ \delta \leq 0$.
- The over-coverage convex function $\Theta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ incurs a penalty of $\Theta(\sigma_{jp})$, for each job j and period p having an over-coverage of σ_{jp} units.

The convex functions help distributing the violations across the problem horizon, in contrast with a linear cost function. The convex function $f(\lambda_e)$ fairly balances the working hours among employees. Also the violation of employee minimum working hours is distributed over all employees instead of being satisfied for almost all employees except one who does not work at all, using the convex function $h(\delta_e)$. Finally, $\Theta(\sigma_{jp})$ favors several periods with one unit of over-coverage over several units of over-coverage for one period. Piecewise linear representations of these convex functions are used in the mixed-integer linear program described in the next section.

The final cost of schedule $\mathcal{S}$ is:

$$c(\mathcal{S}) = \sum_{e \in E_g} f(\lambda_e) + f_{op}(\lambda_{op}) + \sum_{e \in E_g} h(\delta_e) + \sum_{j \in J_g} \sum_{p \in P_g} \Theta(\sigma_{jp}). \tag{6.1}$$

Given the set of jobs J_g and their requirements for the whole horizon, along with the employee set E_g and their shift profiles, the MJ-ESP consists of finding the schedule $\mathcal{S}$ with the lowest cost $c(\mathcal{S})$.

6.2 A mixed-integer program formulation

Next we model the formal MJ-ESP as a mixed-integer program. We first introduce the used variables.

For each shift $s \in S$, a binary variable x_s equals 1 if the shift is selected within the final schedule $\mathcal{S}$, 0 otherwise. For each open shift $s \in S_{op}$ an integer variable y_s takes the value of the number of times the open shift is used within the final schedule $\mathcal{S}$. The value of the over-coverage units for job $j \in J_g$ during period $p \in P_g$ is captured by the integer variable o_{jp}. Let $t_{min,e}$ and $t_{max,e}$ be the desired minimum and maximum employee e working hours, respectively. For each employee $e \in E_g$, let his minimum rest duration between any two successive shifts be $r_{min,e}$. Also we denote the length of a shift s by $|s|$. The proposed mathematical formulation is:

$$\text{Minimize} \sum_{e \in E_g} f\Big(\sum_{s \in S_e} x_s * |s|\Big) \quad + f_{op}\Big(\sum_{s \in S_{op}} y_s * |s|\Big) + \sum_{e \in E_g} h\Big(t_{min,e} - \sum_{s \in S_e} x_s * |s|\Big) + \sum_{j \in J_g} \sum_{p \in P_g} \Theta(o_{jp}) \tag{6.2a}$$

subject to

$$\sum_{\substack{s \in S: \\ j = job(s) \\ p \in Periods(s)}} x_s + \sum_{\substack{s \in S_{op}: \\ j = job(s) \\ p \in Periods(s)}} y_s - \sum o_{jp} = r_{jp} \qquad \forall j \in J_g \text{ and } p \in P_g \tag{6.2b}$$

$$\sum_{s \in S_e} x_s * |s| \leq t_{max,e} \qquad \forall e \in E_g \tag{6.2c}$$

$$\sum_{\substack{s \in S_e \\ day(s) = d}} x_s \leq 1 \qquad \forall e \in E_g \text{ and } d \in D_g \tag{6.2d}$$

$$\sum_{\substack{s \in S_e: \\ \{p_k, \ldots, p_{k+r_{min,e}}\} \\ \cap Periods(s) \neq \emptyset}} x_s \leq 1 \qquad \forall e \in E_g \text{ and } k \in \{1, \ldots, |P_g| - \ r_{min,e}\} \tag{6.2e}$$

$$x_s \in \{0, 1\} \qquad \forall s \in S \tag{6.2f}$$

$$y_s \in \mathbb{Z}_{\geq 0} \qquad \forall s \in S_{op} \tag{6.2g}$$

$$o_{jp} \in \mathbb{Z}_{\geq 0} \qquad \forall j \in J_g \text{ and } p \in P_g \tag{6.2h}$$

The objective function (6.2a) minimizes the sum of the schedule costs. Constraints (6.2b) ensure that the demand in employees is covered for each job and each period, and also compute the values of the over-coverage variables. Constraints (6.2c) impose the maximum working hours for each employee. Constraints (6.2d) restrict each employee to work a maximum of

one shift per day. Constraints (6.2e) ensure that the minimum rest time between each pair of consecutive shifts for each employee is respected. This is accomplished by restricting the assignment of two successive shifts, for a given employee $e \in E_g$, where the difference between the ending time of the first shift and the starting time of the second shift is less than the minimum rest time $r_{min,e}$. Because an employee can work a maximum of one shift per day, this constraint is only concerned with shifts ending within the last hours of a day and shifts starting within the first hours of the next day. Finally, constraints (6.2f), (6.2g) and (6.2h) indicate the domain of each decision variable.

Other constraints like the employee maximum and minimum shift durations, or the employee job-day availability, are handled during the shift enumeration process, i.e., shifts with a duration out of the valid duration range or associated with an employee unavailability period are not enumerated.

This mixed integer linear program is the core model of Kronos Workforce Central (WFC) system. But the WFC software uses some heuristics during shift enumeration and decomposes the program in order to accelerate the solution process while preserving solution quality as much as possible. We cannot present any further details about WFC exact solution process because of confidentiality reasons. WFC will be used twice in our project. First, for each test instance, it will be run to find the best solution it can achieve in its usually allocated computational time. These solutions and computational times will serve as the basic comparative results for testing the proposed matheuristic. Furthermore, WFC will be integrated in our matheuristic for solving the subproblems arising in the large neighborhood search framework.

6.3 Large neighborhood search

Large neighborhood search (LNS) [69] is characterized by the large number of possible neighbor solutions explored in each iteration. In order to overcome the challenge of exploring every single neighbor solution, comparing the costs of all neighbor solutions and choosing the best among them, the neighborhood exploration is replaced by a repair operator in LNS. The repair operator is an algorithm for rebuilding a partially destroyed solution in order to create a higher quality (lower cost) new neighbor solution. Before repairing a solution, part of it must be destroyed first. A destroy operator is an algorithm that chooses part of the current solution to be removed or destroyed. The choice of the destroyed part of the current solution is crucial as the quality of the new repaired solution depends, both equally, on the

destroy operator as on the repair operator. LNS consists of running sequentially a destroy operator then a repair operator, until a stopping criterion is met.

LNS, with its destroy and repair operators, is best suited for the type of problems that can be decomposed into sub-problems having mutually exclusive constraint sets, meanwhile all sub-problems respect the master problem constraints, (Pisinger and Ropke [61]). This decomposition can be found in our MJ-ESP where a sub-problem is the problem of creating the sub-schedule S_{sub} for a subset of the employees $E_{sub} \subset E_g$ for a subset of jobs $J_{sub} \subset J_g$ over a subset of the problem horizon period $P_{sub} \subset P_g$, while maintaining the problem global constraints.

6.3.1 Destroy operator

Sub-scope

A sub-scope is the ordered couple $[JobDays, Employees]$, where $JobDays$ is a set of ordered couples [*job*, *day*], where $job \in J_g$, $day \in D_g$, and $Employees \subset E_g$ is a subset of employees. Let JD_g be the set of all JobDays in the problem.

Given a current solution schedule $\mathcal{S}_{current}$, our destroy operator returns *a list of destroyed sub-scopes*: a list of $[JD_{dest}, E_{dest}]$. A sub-scope $[JD_{dest}, E_{dest}]$, within a destroyed sub-scope list, implies that all shifts $s \in \mathcal{S}_{current}$ assigned to any employee $emp(s) \in E_{dest}$ for a JobDay $[job(s), day(s)] \in JD_{dest}$ are removed from the schedule $\mathcal{S}_{current}$.

Table 6.1 Employee availability per day for both jobs

Employee	day_1	day_2	day_3	day_4	day_5	day_6	day_7
e_1	✓	✓	✓			✓	✓
e_2	✓		✓	✓	✓	✓	
e_3	✓	✓		✓	✓		✓
e_4	✓	✓		✓	✓		✓
e_5		✓	✓	✓		✓	✓

An example

Figure 6.1 presents a non-optimized solution for an instance with two jobs and five employees, $J_g = \{j_1, j_2\}$, $E_g = \{e_1, e_2, e_3, e_4, e_5\}$. This instance is used through the chapter for

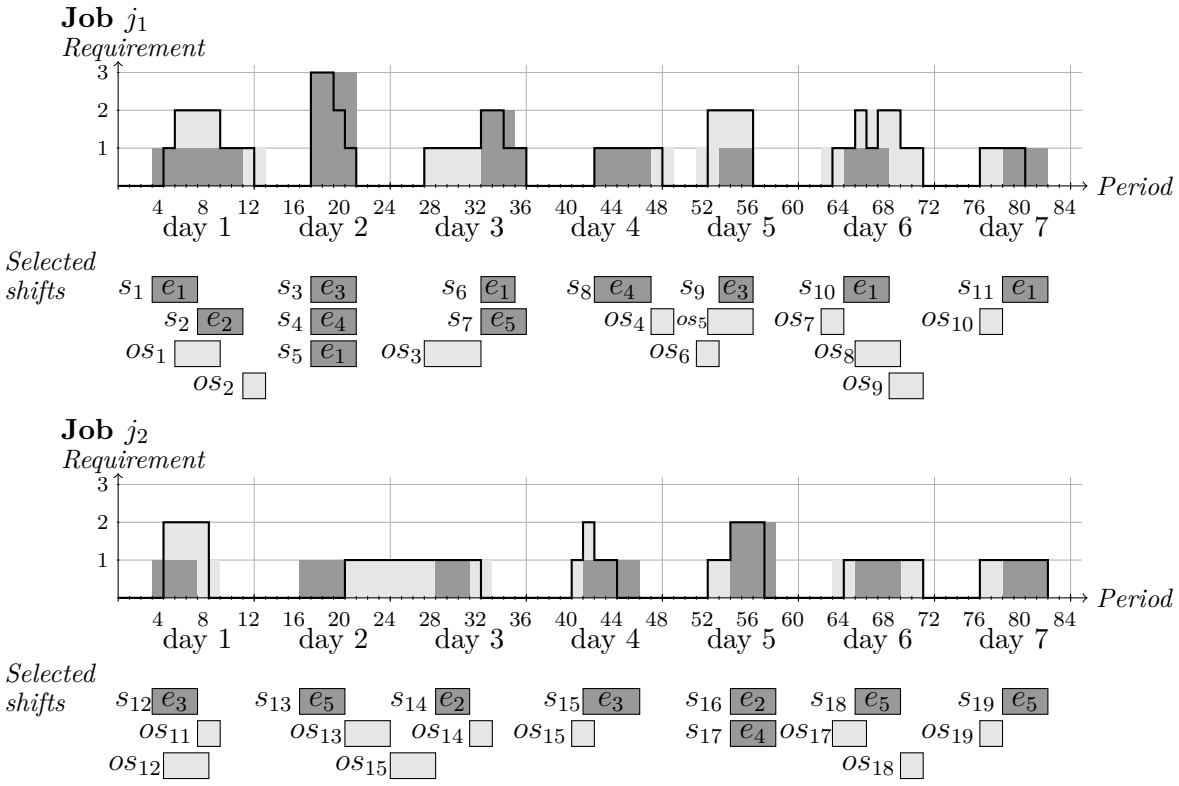

Figure 6.1 Current schedule for the illustrative example.

illustrating the different steps and concepts of our matheuristic.

The time horizon is one week, discretized into 2 hours periods. The shift profile for all employees and for open shifts is as follows: possible shift lengths are 4, 6, 8 and 10 hours. Shifts can start at the beginning of any time period. The minimum rest duration is 10 hours. The maximum employee working hours per week is 40 hours. All employees are qualified for both jobs. The employee time availability is presented in Table 6.1.

In Figure 6.1 each job graph consists of the job requirement curve with the job schedule shifts plotted underneath. Fulfilled requirement by personalized shifts is shaded in dark gray, while fulfilled requirement by open shifts is shaded in light gray.

The current schedule of Figure 6.1 contains 19 personalized shifts and 19 open shifts. The

global JobDay set contains the following elements:

$$JD_g = \{[j_1, day_1], [j_1, day_2], [j_1, day_3], [j_1, day_4], [j_1, day_5], [j_1, day_6], [j_1, day_7],$$
$$[j_2, day_1], [j_2, day_2], [j_2, day_3], [j_2, day_4], [j_2, day_5], [j_2, day_6], [j_2, day_7]\}.$$

An example of a destroyed sub-scope list is: $\{[\{[j_1, day_1]\}, \{e_1, e_2\}], [\{[j_2, day_4]\}, \{e_3\}]\}$. Destroying these sub-scopes leads to the removal of the shifts s_1, s_2 and s_{15} from the current solution.

Destroyed sub-scope choice

At each LNS iteration, the destroyed sub-scopes are the ones that will be repaired. The repair operation, (Section 6.3.2) re-optimizes the destroyed sub-scope schedules. Thus the choice of the sub-scopes to be destroyed must focus on parts of the current schedule causing high costs or penalties.

Three main costs in the objective function map to a weak schedule: cost due to over-coverage, cost of using open shifts, as an open shift can be seen as an under-coverage for low-quality schedules that do not exploit all employees available working hours, and finally penalties for the employee minimum working hours violation. While any employee shift has its own mandatory cost, normal shifts should not interfere with the destroyed sub-scope choice.

Five types of sub-scopes are used in our heuristic.

- *Max-Under-Coverage*: A sub-scope, where its set of JobDays contains exactly one element. This JobDay has the maximum number of period units covered with open shifts within the current solution.
- *Random-Under-Coverage*: A sub-scope, with a single element JobDay set, where the associated JobDay is randomly chosen from the set of all JobDays containing periods covered with open shifts. While *Max-Under-Coverage* helps in the quick convergence of the heuristic, it happens that it may choose a JobDay for which under-coverage cannot be reduced anymore, i.e. an optimal solution will always contain such open shits. The random choice is important because first it identifies important under-covered JobDay requirement, which does not have the highest under-coverage, but meanwhile causes unnecessary cost. Second it helps diversifying the search.

- *Max-Over-Coverage*: A sub-scope with a single element JobDays set, containing the JobDay with the largest over-coverage in the current solution.
- *Random-Over-Coverage*: A sub-scope with a single randomly selected JobDay containing over-covered periods. The random choice in the over-coverage case has the same reasons and effects as the ones previously discussed for the under-coverage case.
- *Min-Hours-Violation*: A list of sub-scopes with employees working less than their preferred minimum working hours per week.

In the first four sub-scopes, the single JobDay is chosen first, then the associated employees are chosen. On the contrary, for the fifth sub-scope construction, the employees for whom the minimum working duration per week is violated are chosen first, then the corresponding JobDays. More details about the size and the construction of the sub-scopes are presented next.

Destroyed neighborhood size

The destroyed neighborhood size has two components: the number of included JobDays and the number of included employees. The destroyed neighborhood size is proportional with the size of the problem, and can be chosen at run time. The problem parameter dp (or $destroyPercent$) indicates the percentage of JobDays, and employees, to be destroyed from the global set JD_g, and from the global employee set E_g, respectively.

Consider a problem instance with n_j jobs, n_e employees over a horizon of n_d days with a destroy percentage dp. Such instance contains a total of $(n_j * n_d)$ JobDays. Thus the number of destroyed JobDays is $\lceil dp * n_j * n_d \rceil$ and the number of selected employees is $\lceil dp * n_e \rceil$, yielding a number of selected employees per JobDay equals to

$$n_{emp/JobDay} = \lceil \frac{\lceil dp*n_e \rceil}{\lceil dp*n_j*n_d \rceil} \rceil.$$

These limits are for controlling the selected neighborhood size and may be violated. This is due to the way the sub-scopes are selected, as described next. For example, if a JobDay is added to the destroyed sub-scope list because it contains under-coverage, it can be selected again because it also contains over-coverage. Also the sub-scope employee selection sometimes includes all employees working on a given JobDay which can violate the expected number of selected employees. On the other hand, our computational experiments show that the neighborhood sizes are typically very close to the targeted ones.

Destroyed sub-scope list construction

Algorithm 5 presents the sub-scope list construction procedure. All included functions are described in details later in the chapter. During each LNS iteration, the destroyed sub-scope list is constructed in two steps

- The first chooses sub-scopes with under-coverage and over-coverage cost.
- The second chooses sub-scopes with employees violating their minimum working hours per week.

Depending on the state of the current solution, one of the two steps' sub-scope can be empty. If both are empty, then the search terminates.
For each of the two steps, the number of JobDays in the constructed sub-scope is limited to $\lceil dp * n_j * n_d \rceil$ JobDays, and the number of employees per JobDay is always $n_{emp/JobDay}$ where applicable.

For the first sub-scope construction step, four sub-scopes, one for each of the four types related to either under-coverage or over-coverage, are selected, forming a *sub-scope block* (lines 7 to 14 in Algorithm 5). Sub-scope blocks are constructed iteratively until the total number of selected JobDays reaches the $\lceil dp * n_j * n_d \rceil$ limit. We define the *sub-scope block* by a sub-scope list constructed following the next sub-scope type order:

1. Max-Under-Coverage.
2. Max-Over-coverage.
3. Random-Under-Coverage.
4. Random-Over-Coverage.

Each of these four sub-scope types mainly consists of one JobDay thus the needed number of selected sub-scope blocks is

$$n_{block} = \lceil \frac{dp * n_j * n_d}{4} \rceil.$$

The idea behind the *sub-scope block* is to iteratively choose sub-scopes of different types until the maximum number of sub-scopes is reached. If we start to choose sub-scopes of one type, until there exists no more of it, the maximum number of sub-scopes can be reached without

Algorithm 5: getMonoThreadSubScopes(*schedule*, *related*)

```
1  Begin
2      List subscopes ← new List;
3      List subscopes_g ← global subscope;
4      Map underCovMap ← intialize the underCovMap for schedule;
5      Map overCovMap ← intialize the overCovMap for schedule;
6      for i = 1, ..., n_block do
7          ss1 ← subScope_underCoverage(underCovMap, false, related, subscopes_g);
8          add ss1 to subscopes;
9          ss2 ← subScope_overCoverage(overCovMap, false, related, subscopes_g);
10         add ss2 to subscopes;
11         ss3 ← subScope_underCoverage(underCovMap, true, false, subscopes_g);
12         add ss3 to subscopes;
13         ss4 ← subScope_overCoverage(overCovMap, true, false, subscopes_g);
14         add ss4 to subscopes;
15     end
16     mergeEmployees(subscopes);
17     if schedule contains Employee Min-working hours violation then
18         ss5 ← getMinWorkingHoursViolationSubscopes(subscopes_g);
19         add ss5 to subscopes;
20     end
21     return subscopes;
22 end
```

creating sub-scopes of different types. Then we will have less interchange opportunities during the sub-scope re-optimization.

After the creation of the sub-scope blocks, if the current solution cost contains a penalty caused by the violation of employee minimum working hours, then an extra sub-scope list of type *Min-Hours-Violation* is constructed (line 18 in Algorithm 5). While the size of this sub-scope is limited by the number of affected employees, the maximum number of added JobDays is $\lceil dp * n_j * n_d \rceil$.

Sub-scope for under-coverage

The choice of a sub-scope of type Max-Under-Coverage or Random-Under-Coverage starts by evaluating the number of missing employees per period for every job that is called, for simplicity, the number of *under-covered* periods. Under-covered periods are time periods, during which the job requirement is (partially) fulfilled by open shifts. Time periods covered with open shifts but causing over-coverage are not considered as under-covered periods. In Figure 6.1, the number of under-covered periods for job j_1 during period 69 is 2. The number

of under-covered periods for the JobDay $[j_1, day_6]$ is 7: 1 under-covered period at period 64, 1 at period 66, 1 at period 68, 2 at period 69, 1 at period 70, and 1 at period 71. Period 63 does not contain any under-coverage. For an LNS iteration, the number of under-covered periods per job per day for the current solution are computed and stored in a map: *underCovMap*.

Two types of sub-scopes with under-coverage can be chosen. Either Max-Under-Coverage, where the chosen $[job, day]$ contains the highest number of under-covered periods, or Random-Under-Coverage, where the chosen $[job, day]$ is randomly picked from the $underCovMap$. The choice of a $[job, day]$ with the highest number of under-covered periods is a greedy choice, where we hope to achieve the highest enhancement after repairing such sub-scope. But sometimes, such under-coverage is unavoidable and appears in an optimal schedule. Thus choosing a random $[job, day]$ containing any number of under-covered periods is mandatory for the search to avoid stagnation.

Under-covered JobDay employee selection

The employee selection for under-coverage sub-scopes looks for employees qualified to work, but not already working, within the current solution for the selected job during the selected day: $E_{qualified}$. We randomly select $n_{emp/JobDay}$ employees from $E_{qualified}$ to be added to the JobDay forming one complete sub-scope.

The employee selection does not consider whether the selected employees already work their maximum working hours within the current schedule or not. This is because we do not want to limit the repair operator within narrow sub-scopes, while the nature of the large sub-scope selection gives the chance for interchanging shifts between employees. So even if an employee already works full time, a shift swap can be beneficial.

Algorithm 6 presents the creation of a sub-scope based on under-coverage. Three parameters are passed to this function: The $UnderCovMap$, a boolean variable $random$ indicating whether the selected JobDay should be randomly chosen or should be the one with the maximum number of under-covered periods, and finally the boolean variable $related$. If $related$ is true, then a related sub-scope is added to the Max-Under-Coverage sub-scope as presented next. On lines 8 and 10, the main sub-scope JobDay is selected, and then removed from the $UnderCovMap$ on line 12. The associated employees are selected on line 13.

Algorithm 6: subScope_underCoverage($underCovMap, random, related, subscopes_g$)

```
1  Begin
2      List subscopes ← empty;
3      if underCovMap is empty then
4          return subscopes;
5      end
6      JobDay jd ← empty;
7      if random then
8          jd ← random JobDay from underCovMap ∩ subscopes_g.JobDays;
9      else
10         jd ← highest under-covered JobDay from underCovMap ∩ subscopes_g.JobDays;
11     end
12     remove jd from underCovMap;
13     chosenEmps ← n_{emp/JobDay} qualified employees from subscopes_g.employees, not
           already working on jd within S_current;
14     subscopes ← add [[jd], chosenEmps];
15     if !random && related then                          // Add related subscope
16         subscopes.lastEntry.employees ← add 4 employees from subscopes_g.employees
               qualified to work at jd;
17         List emps ← all employees within subscopes;
18         for count = 0, ..., 3 do
19             for each employee e ∈ emps do
20                 jd ← JobDay from subscopes_g.JobDays where e is assigned a shift within
                       S_current;
21                 subscopes ← add [[jd], e];
22             end
23         end
24     end
25     return subscopes;
26 end
```

Example continued: For the current solution in Figure 6.1, the $underCovMap$ has the following entries:

$\{[[j_1, day_1], 5]; \quad [[j_1, day_3], 5]; [[j_1, day_4], 1]; [[j_1, day_5], 5]; [[j_1, day_6], 7]; [[j_1, day_7], 2];$
$[[j_2, day_1], 5]; [[j_2, day_2], 4]; \quad [[j_2, day_3], 5]; [[j_2, day_4], 2]; [[j_2, day_5], 2]; [[j_2, day_6], 3]; [[j_2, day_7], 2]\}.$

When constructing the Max-Under-Coverage sub-scope, the JobDay $[j_1, day_6]$ is selected, then deleted from the $underCovMap$. The employees qualified to work for $[j_1, day_6]$ are $\{e_1, e_2, e_5\}$. As e_1 already works for $[j_1, day_6]$, then one employee ($n_{emp/JobDay} = \lceil \frac{5}{2*7} \rceil = 1$)

from $\{e_2, e_5\}$ is randomly chosen, let it be e_2, forming the sub-scope

$$[\{\ [\ j_1, day_6]\ \}, \{e_2\}]$$

which serves as the first entry in the destroyed sub-scope list *subscopeList*.

Max-Under-coverage *related* sub-scope

Destroying large part of a solution enhances the chance of improving the solution using the repair operator. When the number of under- and over-covered periods decreases in the current solution, which means the current solution is getting closer to an optimal one, the size of the destroyed sub-scope is not large enough for enhancements to be quickly achieved. Thus, whenever the LNS metaheuristic stops enhancing the solution, an intensification step is added by adding a "Related sub-scope" to the Max-Under-Coverage, or the Max-Over-Coverage described later, sub-scope for all subsequent iterations. The algorithm is said to stop enhancing the current solution when a repaired solution has the same cost as before being destroyed and repaired.

When the boolean variable *related* is true and the boolean variable *random* is false when calling Algorithm 6, related sub-scopes are added to the current selected sub-scope (line 15). For a Max-Under-Coverage sub-scope $[\{jd\}, E]$, related employees are added first as follows. Four qualified employees, who are not already working for the JobDay jd are added to the employee set E (line 16). Then for each employee $e \in E$, four random JobDays $[j_{e_i}, d_{e_i}]$, for $i = 1, 2, 3$, and 4, where the employee e is assigned a shift in the current solution, are added to the sub-scope (line 21). Adding extra JobDays where the already selected employees are qualified to work gives better opportunities for shifts swapping between employees.

While it seems that the size of the sub-scope list is highly increased after this step, technically, the neighborhood size remains within the chosen destroy percentage range. This is because the related sub-scope is only added after the number of existing under-covered period units is minimized, and the size of the created Max-Under-Coverage sub-scope is small. Also a related sub-scope is only added for the Max-Under-Coverage sub-scope type, not for the random-Under-Coverage sub-scopes.

Example continued: Next we apply these steps to the Max-Under-Coverage sub-scope $[\{\ [\ j_1, day_6]\ \}, \{e_2\}]$ previously created for the illustrative example. The qualified employees

for $[j_1, day_6]$ are only three $\{e_1, e_2, e_5\}$, but e_1 already works for $[j_1, day_6]$, thus only the two employees $\{e_2, e_5\}$ are selected and added to the sub-scope employee set $\{e_2\}$. The sub-scope becomes

$$[\{\ [\ j_1, day_6]\ \}, \{e_2, e_5\}].$$

Given the size of this example we only add one JobDay instead of four for each employee in $\{e_2, e_5\}$. The employee e_2 works for the JobDays $\{\ [j_1, day_1],\ [j_2, day_3],\ [j_2, day_5]\ \}$, $[j_2, day_3]$ is randomly selected. The employee e_5 works for the JobDays $\{\ [j_1, day_3], [j_2, day_2]$, $[j_2, day_6],\ [j_2, day_7]\}$, the JobDay $[j_1, day_3]$ is randomly selected. Finally the sub-scope list $subscopeList$ equals

$$\begin{aligned}&\{[\{\ [j_1, day_6]\}, \{e_2, e_5\}],\\ &\ [\{\ [j_2, day_3]\}, \{e_2, e_5\}],\\ &\ [\{\ [j_1, day_3]\}, \{e_2, e_5\}]\},\end{aligned}$$

which is equivalent to the one entry sub-scope list

$$\{[\{\ [j_1, day_6], [j_2, day_3], [j_1, day_3]\}, \{e_2, e_5\}]\}.$$

Sub-scope for over-coverage

In order to evaluate the over-coverage existing in a current solution, the number of extra employees working for each period is calculated and stored in a map: $overCovMap$. We use the term *the number of over-covered periods* for the total number of extra employees per period for given time frame. In Figure 6.1, job j_1 has 2 over-covered periods at period 21, and a total of 3 over-covered periods for day_2.

Algorithm 7 presents the construction of the sub-scope to be destroyed because of over-coverage. Two types of sub-scopes can be defined using the $overCovMap$, namely, the Max-Over-Coverage and the Random-Over-Coverage sub-scopes. Sub-scope construction starts by either choosing the JobDay with maximum (line 10) or random (line 8) over-coverage from the $overCovMap$. Then the associated employees are chosen after removing the selected JobDay from the $OverCovMap$.

Algorithm 7: subScope_overCoverage($overCovMap, random, related, subscopes_g$)

```
1  Begin
2      List subscopes ← empty;
3      if overCovMap is empty then
4          return subscopes;
5      end
6      JobDay jd ← empty;
7      if random then
8          jd ← random JobDay from overCovMap ∩ subscopes_g.JobDays;
9      else
10         jd ← highest over-covered JobDay from overCovMap ∩ subscopes_g.JobDays;
11     end
12     remove jd from overCovMap;
13     chosenEmps ← All employees from subscope_g.employees working on jd within S_current;
14     subscopes ← add [[jd], chosenEmps];
15     if !random && related then                          // Add related subscope
16         subscopes.lastEntry.employees ← add one employee from subscopes_g.employees
             qualified to work at jd;
17         List emps ← all employees within subscopes;
18         for each employee e ∈ emps do
19             jd ← JobDay from subscopes_g.JobDays where e is assigned a shift within
                 S_current;
20             subscopes ← add [[jd], e];
21         end
22     end
23     return subscopes;
24 end
```

Over-covered JobDay employee selection

After choosing a $[job, day]$ with over-coverage, we start to choose the employee set to complete the sub-scope. The choice of the employees for an over-coverage sub-scope is simple: all employees already working within the chosen JobDay in the current solution are selected (line 13). The employee selection paradigm helps in reducing the number of over-covered periods, if possible, during the repair operation by eliminating, or at least reducing the length, of some shifts. It also helps in better positioning the remaining shifts within the JobDay. We do not restrict the number of chosen employees to the one calculated with the destroy percentage, or choose to destroy only the employees working on shifts that cause the over-coverage, because over-coverage is sometime caused by a wrong position of a shift that does not cause any over-coverage.

Example continued: For the current solution in Figure 6.1, the $overCovMap$ has the

following entries:

$$\{[[j_1, day_1], 1]; [[j_1, day_2], 3]; \quad [[j_1, day_3], 1]; \quad [[j_1, day_7], 2];$$
$$[[j_2, day_1], 1]; [[j_2, day_2], 4]; \quad [[j_2, day_4], 2]; [[j_2, day_5], 2] \quad \}.$$

When constructing the Max-Over-Coverage sub-scope, the JobDay $[j_2, day_2]$ is chosen, then removed from $overCovMap$. Employee e_5 is then selected forming the sub-scope

$$[\{\ [j_2, day_2]\ \}, \{e_5\}].$$

Adding this sub-scope to the previous sub-scope list $subscopeList$ we get

$$\{[\{[j_1, day_6], [j_2, day_3], [j_1, day_3]\}, \{e_2, e_5\}],$$
$$[\{\ [j_2, day_2]\ \}, \{e_5\}]\}.$$

Max-Over-Coverage *related* sub-scope

The Max-Over-Coverage sub-scope construction is enriched with a *related* sub-scope, as an intensification step, once the metaheuristic has stopped enhancing the solution.

For a chosen Max-Over-Coverage sub-scope $[\{\ [jd]\ \}, E]$, a related sub-scope is constructed as follows: One employee $e \notin E$ who is qualified to work for the JobDay jd is chosen. Employees with a shift profile enabling them to work short shifts are prioritized in this step. This priority is applied because shorter shifts fit better with short job requirement whereas longer shifts always result in over-coverage. The chosen employee is added to the current sub-scope employee set E (line 16 in Algorithm 7). Furthermore, several JobDays related to the $[\{\ [jd]\ \}, E]$ sub-scope are added: for each employee $e \in E$, one random JobDay where the employee e is assigned a shift in the current solution is selected (line 19). The updated sub-scope is then added to the sub-scope list.

Example continued: The previously created Max-Over-Coverage sub-scope $[\{\ [j_2, day_2]\ \}, \{e_5\}]$ is updated as follows. One employee $e \notin \{e_5\}$ who is qualified to work for $[j_2, day_2]$ is chosen. Employees $\{e_1, e_3, e_4, e_5\}$ are qualified to work for $[j_2, day_2]$, thus one employee from the set $\{e_1, e_3, e_4\}$ is randomly selected, say e_1. The updated sub-scope equals

$$[\{\ [j_2, day_2]\ \}, \{e_1, e_5\}].$$

Then, for the related JobDays selection: for employee e_1, $[j_1, day_6]$ is randomly selected, and for e_5, $[j_2, day_6]$ is randomly selected. The Max-Over-Coverage sub-scope now looks like

$$[\{\ [j_2, day_2],\ [j_1, day_6],\ [j_2, day_6]\}, \{e_1, e_5\}].$$

Updating the *subscopeList*:

$$\begin{aligned} subscopeList = \{\ &[\{\ [j_1, day_6], [j_2, day_3], [j_1, day_3]\}, \{e_2, e_5\}], \\ &[\{\ [j_2, day_2],\ [j_1, day_6],\ [j_2, day_6]\}, \{e_1, e_5\}]\ \}. \end{aligned}$$

The related sub-scope is not used at the beginning of the metaheuristic because, during the early iterations, it is preferred to concentrate on the reduction of any over or under-covered periods, while the related sub-scopes help enhancing the quality of the schedule by re-arranging the employee shifts. Related sub-scopes are applied after the first time the metaheuristic stops enhancing the solution, until the end of the metaheuristic runs.
While both the Max-Over-Coverage and Random-Over-Coverage sub-scopes are constructed at each iteration, only the Max-Over-Coverage sub-scope can have its related sub-scope constructed. This is done to avoid expanding the constructed sub-scope too much.

Employee merge

After the creation of all *sub-scope blocks*, a post-processing step is accomplished. Employees of the several created sub-scopes are merged to form one set of employees E_{sub}, which replaces all sets of employees in the whole sub-scope list. This step aims to maximize the possible shift interchange among the several chosen sub-scopes. Algorithm 8 presents the employee merge process, and is called in line 16 of Algorithm 5 after the creation of all sub-scope blocks.

Algorithm 8: mergeEmployees(*subscopes*)

```
1 Begin
2 |   Set allEmployees ← new Set;
3 |   for each subscope ∈ subscopes do
4 |   |   add subscope.employees to allEmployees;
5 |   end
6 |   for each subscope ∈ subscopes do
7 |   |   subscope.employees ← allEmployees;
8 |   end
9 end
```

Example continued: Let us assume that the previous sub-scope list *subscopeList* is the one obtained after all sub-scope blocks are created. Applying the employee merge process to the sub-scope list *subscopeList*, we get the updated sub-scope list:

$$\{[\{[j_1, day_6], [j_2, day_3], [j_1, day_3]\}, \{e_1, e_2, e_5\}],$$
$$[\{\ [j_2, day_2], [j_1, day_6], [j_2, day_6]\}, \{e_1, e_2, e_5\}]\},$$

which is identical to the one entry sub-scope list:

$$\{\ [\{\ [j_1, day_6], [j_2, day_3], [j_1, day_3], [j_2, day_2], [j_2, day_6]\}, \{e_1, e_2, e_5\}]\ \}.$$

Kronos WFC solver is the tool optimizing the final created mathematical problem for the destroyed sub-scopes. It is important to know that the solver has a maximum number of enumerated shifts per employee. Consequently, merging the different created sub-scope employees does not affect the computational time enormously but gives the opportunity for effective shift swaps.

Sub-scope for violated employee minimum working hours

Violating the minimum weekly employee working hours is, in general, very costly. In each LNS iteration, if the current solution incurs a minimum employee working hours violation penalty, a *Min-Hours-Violation* sub-scope is constructed and added to the destroyed sub-scope list. The purpose of this sub-scope is to give to the affected employees the chance to be assigned to more shifts, either by covering an under-coverage, replacing another employee, or simply working on a new shift.

Algorithm 9 presents the selection procedure for the sub-scope targeting employee minimum hours violation. The sub-scope selection starts by identifying employees whose minimum working hours is not met *emps* (line 5). Each employee $e \in emps$ qualified jobs J_e are fetched. For each job $j \in J_e$ one random day $d \in D_{e,j}$ is selected. $D_{e,j}$ is the set of days where employee e can work for job j. One sub-scope $[\{\ [\ j, d]\ \}, chosenEmps]$ is constructed and added to the final sub-scope list, where *chosenEmps* consists of the set of all employees working for the job j during day d in the current schedule, as well as the employee e. Sub-scope construction stops when the number of created sub-scopes achieves the maximum limit indicated on line 2 of Algorithm 9, which is the number of all JobDays in the problem multiplied by the problem

parameter *dp*. This limit seems high, but it is mandatory, as in the last iterations when the LNS algorithm starts to converge, the sub-scopes with under-coverage or over-coverage can be empty, thus the size of the minimum hours violation sub-scope must be large enough to improve the solution.

Algorithm 9: getMinWorkingHoursViolationSubscopes($subscopes_g$)

```
1  Begin
2      maxSubscopeCount ← ⌈dp * n_j * n_d⌉;
3      subscopeCount ← 0;
4      subscopes ← new List;
5      emps ← employees from subscopes_g.employees working less than min hours in the
         current schedule; ordered by the number of violated hours;
6      for each employee e ∈ emps do
7          for each job j ∈ J_e do
8              Day d ← random day from D_{e,j};
9              JobDay jd ← [j, d];
10             if jd ∉ subscopes_g.JobDays then          // when using the parallelism
11                 continue;                                          // skip this JobDay
12             end
13             List subscopeEmps ← all employees from subscopes_g.employees working at jd
                 within S_current;
14             subscopeEmps ← add e;
15             subscopes ← add [[jd], subscopeEmp];
16             increment subscopeCount;
17             if subscopeCount ≥ maxSubscopeCount then
18                 return subscopes;
19             end
20         end
21         if subscopeCount ≥ maxSubscopeCount then
22             return subscopes;
23         end
24     end
25     return subscopes;
26 end
```

Example continued: Let us assume that the employee e_4 is working less than his minimum working hours. Constructing the min-hour-violation sub-scope starts by choosing employee e_4. Employee e_4 is qualified to work for both jobs j_1 and j_2. For the job j_1, randomly day day_6 is chosen, forming the new sub-scope $[\ \{[j_1, day_6]\ \}, \{e_4,\ e_1\}]$. We opt for skipping the next sub-scope associated with job j_2 for simplicity. Adding the created sub-scope to the

sub-scope list *subscopeList*, the final sub-scope list becomes:

$$\{[\{\ [j_1, day_6], [j_2, day_3], [j_1, day_3], [j_2, day_2], [j_2, day_6]\}, \{e_1, e_2, e_5\}],$$
$$[\ \{[j_1, day_6]\ \}, \{e_4,\ e_1\}]\}.$$

The final destroyed solution is presented in Figure 6.2. Any shift worked by one of the employees $\{e_1, e_2, e_5\}$ for any JobDay from $\{\ [j_1, day_6], [j_2, day_3], [j_1, day_3], [j_2, day_2], [j_2, day_6]\}$, is removed from the schedule. Similarly, any shift worked by one of the employees $\{e_4,\ e_1\}$ for the JobDay $[j_1, day_6]$ is also removed from the schedule. Open shifts are always removed in the destroyed solution.

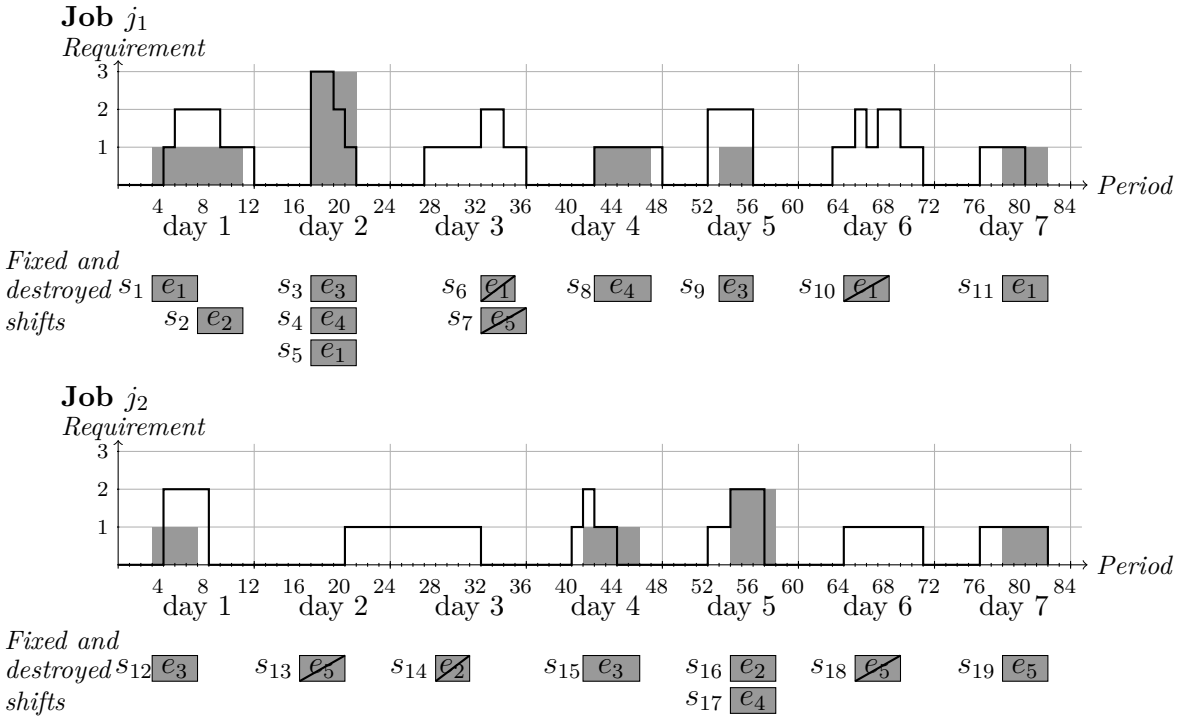

Figure 6.2 The destroyed solution. Destroyed shifts are crossed out. Fixed shifts and their requirement coverage are kept unchanged.

The presented destroy operator is a hybrid operator. It gathers worst-destroy, related destroy and random destroy paradigms. This hybrid nature aims at maximizing each iteration improvement.

The destroyed sub-scope choice procedure terminates by generating the list of sub-scopes to

be destroyed from the current solution, and sending it to the repair operator described next.

6.3.2 Repair operator

A destroyed sub-scope $[JD_{dest}, E_{dest}]$ means that all shifts $s \in \mathcal{S}_{current}$ assigned to any employee $emp(s) \in E_{dest}$ for a JobDay $[job(s), day(s)] \in JD_{dest}$ are removed from the schedule $\mathcal{S}_{current}$. The remaining set of shifts belonging to $\mathcal{S}_{current}$ are preserved in the set of *fixed shifts* S_{fixed}. A set of fixed shifts means that the repair operator is not allowed to change or remove any shift assignment from the fixed set S_{fixed}.
For the destroyed solution of Figure 6.2:

$$S_{fixed} = \{s_1, s_2, s_3, s_4, s_5, s_8, s_9, s_{11}, s_{12}, s_{15}, s_{16}, s_{17}, s_{19}\}.$$

Once a solution is destroyed, it is ready for repair. Our repair operator re-optimizes the destroyed solution through model (6.2) and the Kronos WFC solver, after fixing the x_s variable associated with any shift $s \in S_{fixed}$ to one:

$$x_s = 1 \qquad \forall s \in S_{fixed} \tag{6.3}$$

Furthermore, only shifts associated with the destroyed sub-scopes are enumerated.

6.3.3 Algorithm pseudo-code

Algorithm 10 presents the LNS metaheuristic. The destroyed solution is re-optimized at line 9. Then the new solution cost is compared with the current solution cost (line 10). If an improvement is realized, the new solution is accepted and serves as the current solution for the next iteration. Otherwise, the new solution is not accepted, and we have a failed iteration. The counter $numOfFailures$ is incremented (line 12), and the *related* flag if set to *true* (line 11), in order to add the *related* sub-scope to all subsequent iterations.
The metaheuristic stopping criterion is reaching a total of n_f failed iterations.

Algorithm 10: LNS()

```
1  Begin
2      Boolean related ← false;
3      Int numOfFailures ← 0;
4      Schedule currentSchedule ← getInitialSolution();
5      Schedule newSchedule;
6      List destroyedSubscopes;
7      while numOfFailures ≤ n_f do
8          destroyedSubscopes ← getMonoThreadSubScopes(currentSchedule, related);
9          newSchedule ← reOptimize(destroyedSubscopes, currentSchedule);
10         if newSchedule.cost ≥ currentSchedule.cost then
11             related ← true;
12             increment numOfFailures;
13         else
14             currentSchedule ← newSchedule;
15         end
16     end
17     return currentSchedule;
18 end
```

6.4 Parallel large neighborhood search

Algorithms 6, 7 and 9, for creating the sub-scopes related to under-coverage, over-coverage, and minimum working hour violations, respectively, are the building blocks of our LNS. Algorithms 10 and 5 are the controllers for the one-thread version of the metaheuristic. Next we present how we use Algorithms 6, 7 and 9 in the parallel version of the LNS metaheuristic for solving the MJ-ESP.

6.4.1 Domain decomposition

Our parallel LNS algorithm is categorized as a "Domain decomposition" parallel metaheuristic, and follows the "1C/RS/SPSS" taxonomy category of Crainic [17], i.e. the parallel metaheuristic has *1-C*ontroller, with *R*igid *S*ynchronous communication, uses *S*ame starting *P*oint (initial solution) and *S*ame *S*earch strategy (see section 2.3.3 for more details).

The feasible domain of the MJ-ESP is the global sub-scope. In the parallel LNS with t threads, the destroy operator returns a list of t mutually exclusive sub-scope lists. Then, in each thread, one sub-scope list is repaired with the thread repair operator. Consequently, in each iteration, there is a single destroy operator instance and t repair operator instances, one for each thread.

Algorithm 11: parallelLNS($numThreads$)

```
1  Begin
2      Boolean related ← false;
3      Int numOfFailures ← 0;
4      Schedule currentSchedule ← getInitialSolution();
5      List<Schedule> newSchedulesList;
6      List<Subscopes> destroyedSubscopesList;
7      List<Thread> threads ← initialize numThreads new threads;
8      while numOfFailures ≤ n_f do
9          destroyedSubscopesList ←
             getMultiThreadsSubScopes(currentSchedule, numThreads, related);
10         Check thread sub-scope sizes, merge sub-scopes and update numThreads if
             necessary;
11         for i = 0, ..., numThreads − 1 do
12             newSchedulesList_i ←
                 reOptimize(destroyedSubscopesList_i, currentSchedule) on thread_i;
13         end
14         threads.join();                          // wait until all threads return
15         reset numThreads if it has been changed on line 10;
16         newSchedule ← merge(newSchedulesList);
17         if newSchedule.cost ≥ currentSchedule.cost then
18             related ← true;
19             increment numOfFailures;
20             numThreads ← 1;
21         else
22             currentSchedule ← newSchedule;
23             reset numThreads;
24         end
25     end
26     return currentSchedule;
27 end
```

Algorithm 11 presents the parallel LNS metaheuristic. The metaheuristic starts in the main thread. Then for each iteration, when the destroy operator returns the t disjoint sub-scope lists (line 9), the t sub-scope lists are distributed over the t parallel threads, where each sub-scope list is repaired (line 12). When all threads finish their sub-scope repair, the execution is then returned to the main thread, where the t repaired solutions are merged, forming the new solution (line 16). The new solution cost is compared with the current solution cost, and is chosen as the next iteration current solution if an improvement is achieved. Otherwise the number of failed iterations is incremented and the *related* flag is set to true.

Parallel LNS Algorithm 11 is analogous to the one-thread LNS Algorithm 10. Line 8 creates one sub-scope for the one-thread version Algorithm 10, which is replaced by the line 9 in the parallel Algorithm 11, where several sub-scopes are returned, one for each thread. Then the repair operation is directly executed on the main thread for the one-thread version (line 9), while this is replaced by lines 11-16 in the parallel LNS, where all threads optimize their sub-scopes and return their new local solutions which are then merged to form a new complete solution (line 16). Next each new parallel component is presented.

6.4.2 Multi-thread destroy operator

First, it is important to note that the multi-thread destroy operator runs on the main thread, but creates t disjoint sub-scopes to be next optimized in parallel. The different thread destroyed sub-scopes must be mutually exclusive so when they are re-optimized and merged, the final solution is always feasible. The feasibility of the merged solution is maintained.

The main idea behind the multi-thread destroyed sub-scope selection is to forbid the choice of either a JobDay or an employee already selected within another thread sub-scope. This is done by maintaining a different *domain* for each thread, acting as the thread global sub-scope. All thread domains are initialized to the problem global sub-scope, and updated every time a destroyed thread sub-scope is updated. Each time a sub-scope $subscopes_i$ is added for thread t_i destroyed sub-scope list, all $JobDays \in subscopes_i$ are removed from the thread domain of all other threads in order to forbid these other threads from choosing any of these JobDays in the future. Similarly, all $employees \in subscopes_i$ are removed from the thread domain of all other threads in order to disallow the choice of any of these employees for another thread sub-scope.

We illustrate this procedure for a 2-thread LNS iteration using the current solution in Figure 6.1. The thread domains $threadDomain_1$ and $threadDomain_2$ are initialized to the global sub-scope, which we formulate as follows:

$$\{[\ \{[j_1, day_1],\ [j_1, day_2],\ [j_1, day_3],\ [j_1, day_4],\ [j_1, day_5],\ [j_1, day_6],\ [j_1, day_7],$$
$$[j_2, day_1],\ [j_2, day_2],\ [j_2, day_3],\ [j_2, day_4],\ [j_2, day_5],\ [j_2, day_6],\ [j_2, day_7]\},\ \{e_1, e_2, e_3, e_4, e_5\}\ ]\}$$

Let the first thread created sub-scope be:

$$[\ \{[j_1, day_1]\ \}, \{e_1, e_2\}].$$

The $threadDomain_1$ remains unchanged, while the $threadDomain_2$ becomes:

$$\{[\{\quad [j_1, day_2],\ [j_1, day_3],\ [j_1, day_4],\ [j_1, day_5],\ [j_1, day_6],\ [j_1, day_7],$$
$$[j_2, day_1],\ [j_2, day_2],\ [j_2, day_3],\ [j_2, day_4],\ [j_2, day_5],\ [j_2, day_6],\ [j_2, day_7]\},\ \{e_3, e_4, e_5\}\ ]\}$$

Afterward, any sub-scope created for the second thread is guaranteed to be disjoint from the first thread sub-scope.

While the parallel thread domains are maintained disjoint with the previously mentioned procedure, the thread sub-scopes are chosen in the same way as the one-thread version (Algorithm 5), but each step is repeated t times, where t is the number of threads. Algorithm 12 presents the multi-thread sub-scope selection process. The algorithm starts by initializing all thread domains to the problem global domain (line 6). Then, for n_{blocks} times, each of the four sub-scope types (*Max-Under-Coverage*, *Max-Over-Coverage*, *Random-Under-Coverage*, and *Random-Over-Coverage*) is created for the t threads. Algorithm 13, which is called in lines 12 and 16, creates the *Max-Under-Coverage* and *Random-Under-Coverage* sub-scopes for t threads. Algorithm 14, which is called in lines 14 and 18, creates the *Max-Over-Coverage* and *Random-Over-Coverage* sub-scopes for t threads.
In the one-thread sub-scope creation algorithm (Algorithm 5, line 16), all sub-scope employees are merged after the n_{blocks} sub-scope blocks are created. Similarly, in the multi-thread version (Algorithm 12, line 22), for every thread sub-scope, all employees are merged. Finally, Algorithm 15 which is called in line 25, creates the *Min-Hours-Violation* sub-scope for t threads if the current solution cost contains a minimum working hours violation penalty.

All the algorithms (Algorithm 13 which creates the under-coverage sub-scopes for all the threads, Algorithm 14 which creates the over-coverage sub-scopes for all the threads, and Algorithm 15 which creates the minimum hours violation sub-scope for all the threads) take care of maintaining the disjoint property of all thread sub-scopes. The three algorithms follow the same pattern: The algorithms iterate over the multiple threads (line 3). Then, in line 4 the sub-scope creation algorithms - 6, 7 or 9 - are called like in the one-thread version, but the thread domain is sent as a parameter serving as the thread global sub-scope. After each thread sub-scope creation, the JobDays and employees appearing in the created sub-scope

Algorithm 12: getMultiThreadSubScopes(*schedule*, *related*, *numThreads*)

```
1  Begin
2      List<subscopes> threadsSubscopes ← new List;
3      List subscopes_g ← global subscope;
4      List<subscopes> threadsDomains ← new List;
5      for i = 0, ..., numThreads − 1 do
6          threadsDomains_i ← subscopes_g;
           // each thread domain is initialized to the global domain
7          threadsSubscopes_i ← new List;
8      end
9      Map underCovMap ← intialize the underCovMap for schedule;
10     Map overCovMap ← intialize the overCovMap for schedule;
11     for n_blocks times do
12         ss_1 ←threadsSubscopes_underCoverage(numThreads, false, related);
13         update threadsSubscopes with ss_1;
14         ss_2 ←threadsSubscopes_overCoverage(numThreads, false, related);
15         update threadsSubscopes with ss_2;
16         ss_3 ←threadsSubscopes_underCoverage(numThreads, true, false);
17         update threadsSubscopes with ss_3;
18         ss_4 ←threadsSubscopes_overCoverage(numThreads, true, false);
19         update threadsSubscopes with ss_4;
20     end
21     for i = 0, ..., numThreads − 1 do
22         mergeEmployees(threadsSubscopes_i);
23     end
24     if schedule contains employee minimum working hours violation then
25         ss_5 ←threadsSubscopes_minHoursViolation(numThreads);
26         update threadsSubscopes with ss_5;
27     end
28     return threadsSubscopes;
29 end
```

are removed from the other thread domains (lines 7 and 8).

Algorithm 13: threadsSubscopes_underCoverage($numThreads, random, related$)

```
1  Begin
2      List<subscopes> threadsSubscopes ← new List;
3      for i = 0, ..., numThreads − 1 do
4          threadsSubscopes_i ← subScope_underCoverage(underCovMap,
                                    random, related, threadsDomains_i);
5          for j ← 0, ..., numThreads − 1 do
6              if j != i then
7                  remove all JobDay ∈ subscopes from threadsDomains_j;
8                  remove all employee ∈ subscopes from threadsDomains_j;
9              end
10         end
11     end
12     return threadsSubscopes;
13 end
```

Algorithm 14: threadsSubscopes_overCoverage($numThreads, random, related$)

```
1  Begin
2      List<subscopes> threadsSubscopes ← new List;
3      for i = 0, ..., numThreads − 1 do
4          threadsSubscopes_i ← subScope_overCoverage(overCovMap,
                                    random, related, threadsDomains_i);
5          for j ← 0, ..., numThreads − 1 do
6              if j != i then
7                  remove all JobDay ∈ subscopes from threadsDomains_j;
8                  remove all employee ∈ subscopes from threadsDomains_j;
9              end
10         end
11     end
12     return threadsSubscopes;
13 end
```

Sub-scope creation order

Sub-scope blocks are created one by one for all threads as follows:

- One sub-scope with Max-Under-Coverage is created for each thread.
- One sub-scope with Max-Over-Coverage is created for each thread.
- One sub-scope with Random-Under-Coverage is created for each thread.

Algorithm 15: threadsSubscopes_minHoursViolation($numThreads$)

```
1  Begin
2      List<subscopes> threadsSubscopes ← new List;
3      for i = 0, ..., numThreads − 1 do
4          threadsSubscopes ← getMinWorkingHoursViolationSubscopes(
             threadsDomains_j);
5          for j ← 0, ..., numThreads − 1 do
6              if j != i then
7                  remove all JobDay ∈ subscopes from threadsDomains_j;
8                  remove all employee ∈ subscopes from threadsDomains_j;
9              end
10         end
11     end
12     return threadsSubscopes;
13 end
```

- One sub-scope with Random-Over-Coverage is created for each thread.

This order allows the JobDays containing the highest amount of under/over-covered periods to be distributed over the several threads and ensures that every thread gets a significant number of employees.

Intensification and diversification

The number of used threads t is a parameter set at the beginning of the metaheuristic (Algorithm 11). When no improved solution is found in an iteration, an intensification/diversification step is taken, by swapping the number of used threads between one and t. The intensification occurs when the number of threads is switched from t to one (Algorithm 11, line 20), giving the opportunity to create larger sub-scopes for the single-thread. On the other hand, diversification arises when the number of threads is set back to t (line 23).

Another intensification step is taken after the first iteration without improvement, where the related sub-scope is allowed for the Max-Under-Coverage or the Max-Over-Coverage sub-scopes for subsequent iterations (Algorithm 11, line 18).

Number of threads and neighborhood size

As the number of under/over-covered periods and the violated employee minimum working hours are minimized, the sizes of each thread neighborhood is also minimized. When a thread neighborhood is too small, the repair operation can lead to poor or no improvement.

In order to always maintain a large neighborhood size for each thread, a post-processing is accomplished after creating the thread neighborhood as follows. The thread neighborhoods are increasingly ordered by the number of JobDays in each thread neighborhood. Then we iterate over each thread neighborhood, if the number of a thread neighborhood JobDays is less than 10% of the size of the global JobDays set JD_g, this thread neighborhood is merged with the next thread neighborhood, forming a wider new neighborhood, and decreasing the number of threads by one. This is repeated until the number of JobDays in all thread neighborhoods is at least 10% of the total problem JobDays or only one thread is remaining (Algorithm 11, line 10).

While this post-processing minimizes the number of parallel threads for the current iteration, it maintains the remaining threads useful and powerful.

6.5 Computational experiments

We validate the parallel LNS algorithm for the MJ-ESP by running it for several datasets with several initial solutions. Both single-thread and multi-thread versions are tested and compared. This section presents the test environment and analyzes the results. The section is organized as follows. Section 6.5.1 introduces the used datasets and their different initial solutions. Section 6.5.2 presents the experiment parameter settings. We conduct a sensitivity analysis for the LNS destroy percentage in Section 6.5.3, followed by an initial solution analysis in Section 6.5.4. Then the results for the single-thread and the multi-thread LNS algorithms are discussed in Section 6.5.5.

6.5.1 Datasets and initial solutions

We have tested the LNS algorithm on seven datasets provided by our industrial partner. Each dataset is characterized by its number of jobs and number of employees. The seven datasets are: 7J_37E, 8J_94E, 5J_25E, 5J_50E, 7J_74E, 10J_50E and 14J_74E. A dataset name xJ_yE refers to a dataset with x jobs and y employees. For all datasets, the planning horizon is one week, divided into 15-minute periods. Note also that the employees in each dataset have different shift profiles. For each dataset, Table 6.2 reports the cost of the best solution found by the WFC solver using a standard configuration, as well as the total computational time. These costs and times serve as base values for comparison purposes with the results obtained by the proposed LNS metaheuristic.

Kronos WFC solver incorporates an initial solution builder. The builder creates, in general,

Table 6.2 Best solution cost and computational time (in seconds) using the WFC solver.

Dataset	$Cost_{WFC}$	$Time_{WFC}$
5J_25E	41655	97
5J_50E	83420	504
7J_37E	50475	16
7J_74E	101130	170
8J_94E	130860	47
10J_50E	83170	814
14J_74E	100830	486

low-quality solution significantly fast. These initial solutions will be referred to by $IS-WFC$ in the following result presentation. Any test computational time presented next includes the creation time of the initial solution.

In order to test the metaheuristic performance on different scenarios for each dataset, four other initial solutions are created and saved. These solutions have different properties: different number of under-covered hours, over-covered hours and violated employee minimum working hours. The initial solutions creation algorithm is a sequential optimization of different disjoint sub-scopes of the dataset. The size of the sub-scopes is manually adjusted depending on each dataset size. For each dataset, the four created initial solution names are $IS-1$, $IS-2$, $IS-3$ and $IS-4$.

Table 6.3 presents some statistics for the different initial solutions for each dataset. The last four columns in this table provide:

Cost: The initial solution cost.

U/C hours: The number of hours in the initial solution, where the employee demand is covered by open shifts, which is considered as under-coverage.

O/C hours: the number of over-covered hours.

VEM hours: the number of violated employee minimum working hours.

6.5.2 Experimental setting

Experiments were executed on a computer with 4 Intel Core 3.40 GHz CPUs and 32 GB RAM. The Kronos WFC solver uses Xpress 8.4. MILP solver.

Table 6.3 Statistics of the initial solutions.

Dataset	Initial solution	Cost	U/C hours	O/C hours	VEM hours
5J_25E	IS-1	802304	263.3	205.8	126.9
	IS-2	984753	345.3	206.0	165.6
	IS-3	516566	21.3	269.3	0.2
	IS-4	899992	214.0	276.3	59.0
	IS-WFC	41815	0.0	0.0	14.5
5J_50E	IS-1	3242489	511.5	435.5	284.9
	IS-2	2196011	30.0	535.5	0.0
	IS-3	2665373	772.5	372.8	439.8
	IS-4	2124083	554.0	351.0	349.8
	IS-WFC	1020900	0.0	0.0	340.4
7J_37E	IS-1	1233935	316.8	118.0	214.3
	IS-2	168301	2.0	226.0	4.3
	IS-3	684269	280.0	157.3	114.3
	IS-4	840879	234.8	154.0	160.0
	IS-WFC	1195357	349.3	0.0	260.0
7J_74E	IS-1	2119840	562.8	236.0	367.3
	IS-2	601989	50.5	435.8	23.3
	IS-3	2181746	593.8	300.5	457.0
	IS-4	2112337	511.5	259.8	445.0
	IS-WFC	2884834	608.8	50.5	686.3
8J_94E	IS-1	3620141	935.5	53.0	760.0
	IS-2	894538	54.5	689.8	0.0
	IS-3	599809	19.8	581.8	0.0
	IS-4	2846552	951.3	162.5	593.3
	IS-WFC	3552375	1174.0	0.0	795.5
10J_50E	IS-1	3762348	639.0	256.8	593.5
	IS-2	1404462	42.8	668.3	0.0
	IS-3	2428716	676.3	259.8	495.7
	IS-4	1824477	553.8	361.8	342.7
	IS-WFC	999070	0.0	0.0	340.4
14J_74E	IS-1	2709570	460.3	47.8	611.0
	IS-2	321165	0.0	447.3	11.8
	IS-3	1803348	565.8	315.5	309.0
	IS-4	2125781	562.8	192.5	477.3
	IS-WFC	2964876	626.0	0.0	701.8

In all tests, the metaheuristic stopping criteria is having six iterations without enhancement, $n_f = 6$. While this number of failed iterations before termination is small compared to other metaheuristics, which can reach hundreds of iterations, our LNS iterations are time consuming with substantial improvements observed in most of them. Thus six non-improving iterations indicate that improvements become hard to achieve, and it is better to stop the metaheuristic in order to save considerable computational time. Other datasets with different characteristics may need to calibrate the parameter n_f.

For each dataset and each initial solution, we conducted ten different runs with the LNS algorithm. Each of the ten runs has a different seed for the random number generator used for the sub-scope construction. The final result is presented as the average cost gain and computational time gain compared to the results obtained by the standard WFC solver. Denote by $Cost_{LNS}$ and $Time_{LNS}$ (resp. $Cost_{WFC}$ and $Time_{WFC}$) the cost of the best solution found by the LNS algorithm (resp. WFC solver) and its total computational time. The cost gain is calculated as:

$$Gain_{cost} = 100 \ * \ \frac{Cost_{WFC} \ - \ Cost_{LNS}}{Cost_{WFC}}\%$$

whereas the time gain:

$$Gain_{time} = 100 \ * \ \frac{Time_{WFC} \ - \ Time_{LNS}}{Time_{WFC}}\%.$$

Table 6.4 Average cost gain (%) and time gain (%) for different destroy percentages, using one thread.

	Destroy percentage									
	10%		20%		30%		40%		50%	
Dataset	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*
5J_25E	2.1	92.4	2.2	84.3	2.3	82.4	2.2	82.4	2.2	82.1
5J_50E	0.2	91.6	1.6	86.6	1.5	80.3	1.4	79.2	1.5	78.8
7J_37E	1.9	54.3	1.9	48.8	2.1	48.0	2.1	49.2	2.2	42.0
7J_74E	0.0	90.0	1.8	89.2	2.2	88.4	2.3	88.0	2.2	87.4
8J_94E	-0.6	13.5	-0.3	16.1	-0.4	19.7	0.1	21.0	-0.3	20.9
10J_50E	1.2	94.8	1.0	8.8	0.8	84.2	0.8	79.0	0.7	80.1
14J_74E	2.3	94.6	2.4	92.4	2.5	92.2	2.5	91.6	2.5	90.2
Average	*1.0*	*75.9*	*1.5*	*72.2*	*1.5*	*70.8*	*1.6*	*70.1*	*1.6*	*68.8*

6.5.3 Destroy percent analysis

In this section, we perform a sensitivity analysis on the value of the destroy percentage used in the destroy operator of the single-thread LNS metaheuristic. Table 6.4 displays the average cost and time gains for several destroy percentages (10%, 20%, 30%, 40%, and 50%). For each dataset, the average is over 10 runs for each of the five initial solutions ($IS-1, IS-2, IS-3, IS-4$, and $IS-WFC$). The last line in the table presents the cost and time gain averages over all instances.

Low destroy percentage value creates smaller neighborhoods to be optimized, thus less improvement per iteration. Consequently, more iterations are needed before the LNS heuristic convergence. This is observed in Table 6.5, which shows the average number of iterations for the LNS heuristic with each destroy percentage.

In Table 6.4, the results for the 10% destroy percentage are the worst among the other results, but with an average of only 0.6% or less degradation on the cost gain with respect to the higher destroy percentage results, and with an average of 4% or more computation time gain than the other destroy percentage results. For higher destroy percentages, we remark a stabilization on the average cost gain. This is explained by the way the sub-scope selection algorithms use the destroy percentage parameter: the destroy percentage is only used as an upper bound for the selected sub-scopes. Smaller sub-scopes are created when there is no more possible sub-scopes to choose from. For example, if within a solution only one JobDay contains over-coverage, then the function constructing the Max-Over-Coverage sub-scope will return a sub-scope list of size one, and the Random-Over-Coverage sub-scope function will return an empty list, even if larger lists are expected.

Table 6.5 Average number of iterations for different destroy percentages, using one thread.

	Destroy percentage				
Dataset	10%	20%	30%	40%	50%
5J_25E	20.0	15.0	13.2	13.4	12.4
5J_50E	23.2	15.8	14.8	13.0	13.2
7J_37E	15.4	12.6	11.0	10.6	10.0
7J_74E	20.0	16.4	14.4	13.4	12.8
8J_94E	31.6	25.8	22.0	19.8	19.8
10J_50E	23.2	15.8	13.8	13.8	13.2
14J_74E	20.0	15.6	13.0	12.4	11.0

Results in Table 6.4 proves that the LNS heuristic with different destroy percentage converges in a timely manner. Smaller destroy percentages give faster lower quality solutions while higher ones give better solution costs but slightly slower. This gives the opportunity to the user to choose the best destroy percentage depending on the used dataset size. Smaller destroy percentages are preferred with very large datasets, higher destroy percentages are better for medium size datasets. For the remaining computational experiment sections, we use the LNS heuristic with 40% destroy percentage, as it shows the best average cost and computational time gains with respect to the other percentages.

6.5.4 Initial solution analysis

The initial solution analysis tests the LNS heuristic behavior when starting with different solution qualities. Table 6.6 displays the LNS heuristic results for the different initial solutions. The 2^{nd} and 3^{rd} columns show the dataset best solution and computational time using the WFC solver. The 5^{th} column presents the initial solution cost. The 6^{th} and 7^{th} columns display the average optimized solution and computational time, respectively, over 10 runs, using the single-thread LNS heuristic with 40% destroy percentage. Finally the cost and computational time gains are listed in the 8^{th} and 9^{th} columns. For each dataset, the best results are highlighted in bold.

Table 6.6 shows that the LNS heuristic converges similarly with different starting solution qualities for most datasets. Negative gains are obtained only for dataset 8J_94E and the initial solution ($IS-1$). Initial solution $IS-1$ cost is very high, but the $IS-WFC$ initial solution for the same dataset has approximately the same cost as $IS-1$, which shows the independence between the $IS-1$ negative gain (or gap) and the starting solution cost. Instead, it was remarked that during the last iterations for the $IS-1$ optimization, a special sub-scope combination would have been needed to allow some shift swaps and reduce the schedule cost, which was not accomplished before achieving the stopping criterion.

From these results, we observe that the quality of the initial solution is not correlated with the obtained results. Indeed, in certain cases, low-quality initial solutions yield the best average cost gains: For dataset 7J_37E, the best average cost is obtained from the worst initial solution $IS-WFC$. Consequently, for the next LNS heuristic result analysis, we will present results for the initial solution $IS-WFC$. While $IS-WFC$ did not show the best average gains with respect to other initial solutions, we prefer to use it as it is calculated on

Table 6.6 Initial solutions analysis.

Dataset	$Cost_{WFC}$	$Time_{WFC}$	IS-name	IS-Cost	Average $Cost_{LNS}$	Average $Time_{LNS}$	$Gain_{Cost}$ (%)	$Gain_{Time}$ (%)
			IS-1	766319	40745	17.3	2.2	82.1
			IS-2	953673	40751	22.5	2.2	76.7
5J_25E	41655	96.5	IS-3	462251	40714	**14.5**	2.3	84.9
			IS-4	856822	**40711**	16.2	2.3	83.2
			IS-WFC	41815	40719	14.6	2.2	84.8
			IS-1	3168179	82432	124.9	1.2	75.2
			IS-2	2086811	**81946**	160.7	1.8	68.1
5J_50E	83420	503.9	IS-3	2610488	82154	82.4	1.5	83.6
			IS-4	2057393	82156	**73.0**	1.5	85.5
			IS-WFC	1020900	82417	82.3	1.2	83.7
			IS-1	1191964	49365	7.4	2.2	52.5
			IS-2	100965	49347	**5.5**	2.2	64.4
7J_37E	50475	15.6	IS-3	636388	49344	8.5	2.2	45.6
			IS-4	791378	49818	8.8	1.3	43.4
			IS-WFC	1195357	**49284**	9.4	2.4	39.8
			IS-1	2044225	**98271**	18.7	2.8	89.0
			IS-2	483654	100564.5	**13.7**	0.6	91.9
7J_74E	101130	170.4	IS-3	2104121	98547	25.8	2.6	84.9
			IS-4	2032222	98458.5	21.4	2.6	87.5
			IS-WFC	2884834	98304	22.9	2.8	86.6
			IS-1	3545831	134355	54.7	-2.7	-17.1
			IS-2	729163	**128619**	19.9	1.7	57.5
8J_94E	130860	46.7	IS-3	438829	128718	**16.7**	1.6	64.2
			IS-4	2766617	130152	43.9	0.5	6.1
			IS-WFC	3552375	130621.5	43.6	0.2	6.6
			IS-1	3706413	82458	307.2	0.9	62.3
			IS-2	1288062	82511	130.2	0.8	84.0
10J_50E	83170	814.0	IS-3	2374836	**82413**	164.0	0.9	79.9
			IS-4	1757127	82440	**115.6**	0.9	85.8
			IS-WFC	999070	82763	138.6	0.5	83.0
			IS-1	2636070	**98268**	46.2	2.5	90.5
			IS-2	196080	98448	**25.1**	2.4	94.8
14J_74E	100830	485.7	IS-3	1720113	98298	46.6	2.5	90.4
			IS-4	2049746	98400	43.3	2.4	91.1
			IS-WFC	2964876	98310	44.2	2.5	90.9

run time with the LNS execution.

6.5.5 Single-thread and multi-thread LNS results

Using a destroy percentage of 40% and the $IS-WFC$ initial solution, both single-thread and parallel LNS heuristics with up to 6 parallel threads have been tested. Relative cost and time gains are given in Table 6.7, where the last line reports averages over all datasets.

Table 6.7 Average cost gain (%) and time gain (%) for single-thread execution and different number of parallel thread execution. Average cost and time is over 10 runs using the $IS-WFC$ initial solution and 40% destroy percentage.

	1 Thread		2 Threads		3 Threads		4 Threads		5 Threads		6 Threads	
Dataset	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*	*Cost*	*Time*
5J_25E	2.2	84.8	1.8	75.0	2.1	73.1	2.0	74.7	2.0	73.0	1.9	75.5
5J_50E	1.2	83.7	1.8	86.1	1.9	87.0	1.5	86.5	1.2	78.3	1.7	83.8
7J_37E	2.4	39.9	2.4	64.7	2.4	58.3	2.2	70.1	2.1	68.0	2.3	74.9
7J_74E	2.8	86.6	1.5	91.6	-2.0	92.5	2.5	93.3	2.5	93.5	2.6	93.5
8J_94E	0.2	6.7	1.2	27.7	0.9	46.5	0.8	51.8	-0.4	40.9	0.6	49.5
10J_50E	0.5	83.0	0.8	82.4	1.1	88.2	1.0	90.1	0.9	88.4	1.3	89.6
14J_74E	2.5	90.9	2.4	92.5	2.4	96.4	2.4	95.4	2.2	96.2	2.3	96.4
Average	*1.7*	*67.9*	*1.7*	*74.2*	*1.3*	*77.4*	*1.8*	*80.3*	*1.5*	*76.9*	*1.8*	*80.4*

Compared to the single-thread behavior, the parallel metaheuristic succeeds to maintain a high-level performance for the different levels of parallelism. For the cost gain, the best gain is achieved by both the four-thread and six-thread versions. Tables 6.8 and 6.9 report the average number of re-optimized JobDays and employees, respectively, on each thread over all iterations. These tables show that the size of the neighborhoods decrease with the increase on the number of parallel threads, which is expected. This decrease in the re-optimized neighborhood size enhances the computational time. The six-thread LNS version is 18% faster than the single-thread version.

These results show that for very large instances, when the single-thread execution starts to be challenging, multi-threading becomes a safe alternative, guaranteeing a high solution quality. Both parameters t and dp can be changed by the user before launching our proposed parallel LNS heuristic, giving the user the control over the number of parallel threads and the used destroy percentage, respectively. This helps the user choose the best values depending on the used dataset size.

Table 6.8 Average number of JobDays optimized per iteration and thread, for different number of threads using 40% destroy percentage and $IS-WFC$ initial solution.

Dataset	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads
5J_25E	8.3	8.2	7.7	8.0	7.8	7.9
5J_50E	10.3	9.7	9.1	8.6	8.5	8.3
7J_37E	11.0	11.5	10.0	8.4	7.3	6.4
7J_74E	13.6	13.7	10.9	9.6	8.7	7.6
8J_94E	13.1	14.5	11.9	10.7	10.7	9.1
10J_50E	20.7	17.0	15.7	14.8	15.3	14.9
14J_74E	16.8	25.0	21.6	19.4	16.5	15.0
Average	*13.4*	*14.2*	*12.4*	*11.4*	*10.7*	*9.9*

Table 6.9 Average number of employees optimized per iteration and thread, for different number of threads using 40% destroy percentage and $IS-WFC$ initial solution.

Dataset	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads
5J_25E	16.7	14.8	15.0	14.8	14.8	14.6
5J_50E	25.4	20.9	18.5	17.1	15.6	15.5
7J_37E	13.4	11.1	9.3	9.3	7.4	7.0
7J_74E	21.6	14.9	13.4	12.0	10.7	9.8
8J_94E	25.8	23.1	19.9	17.2	16.9	15.1
10J_50E	21.1	18.7	17.1	17.6	15.4	16.6
14J_74E	21.8	19.4	16.6	14.4	14.3	12.2
Average	20.8	17.6	15.7	14.6	13.6	13.0

CHAPTER 7 GENERAL DISCUSSION

In this thesis, we dealt with the challenge of creating employee schedules for large problem instances. Even if personnel scheduling problems have been widely studied in the literature, most solution methods can be classified in two categories. The first are the algorithms addressing the problems involving only the five, or less, shift types per day: Day, Evening, Night, 12-hour a.m. and 12-hour p.m.. In this case, the resulting problems are relatively small when compared with our problems where each employee shift profile contains dozens of different shift types (several shift starting times and several shift lengths). The second category includes the algorithms solving the problem in two steps: a shift allocation step where anonymous shifts fulfilling the demand are created, followed by an employee shift assignment step. We tested this "shift allocation-shift assignment" approach at the beginning of our research, but it was not suitable for the special problem properties we deal with. Thus we did not present it as it did not yield interesting results.

Consequently we needed to explore and study new, out of the box, solution methods. For the ESP-IDT we first studied the two-phase heuristic proposed by Munezero [57]. The first phase optimizes each department employee schedules separately. The second phase uses remaining employee hours to perform any possible transfers. Phase-one department decomposition leads to a loss in the needed inter-department transfer information. To overcome this loss of needed transfer information, we developed a three-phase heuristic MP-DH that starts by analyzing the instance data and building for each department its own inter- and intra-department requirement curves. To reduce computational times, the second phase identifies which departments can provide employees to cover these transfer requirements. Finally, the third phase optimizes each department employee schedules, covering both the department internal and transfer requirement. MP-DH rendered the very large instances solvable in a timely manner, and in shorter computational times compared to Dahmen et al. [22].

The MP-DH third phase solves a ESP-DIDT MILP. Controlling the MILP computational time is a challenge especially for departments with a large number of employees. The semi-anonymous heuristic, presented in Chapter 5, reduces the optimized MILP size but does not reduce the computational time while maintaining accepted solution quality. The main drawback of the SA heuristic is forcing the employee decomposition even when it is not needed. Thus an intelligent heuristic, controlling when to decompose the problem to avoid long com-

putational times, and when not to decompose it in order to maintain solution quality, is mandatory. In this regard, we developed the hybrid heuristic which dynamically decides when to decompose the department employee set and when to optimize all employee schedules. *HH* is able to reduce up to 87.4% of the MILP computational time while maintaining an acceptable solution quality degradation of 3.1%.

The development of parallel metaheuristics is a rapidly evolving area of research, showing success for diverse applications. Multi-start parallel metaheuristic is the most used parallelization method. The multi-start metaheuristic, whether cooperative or not, runs in parallel several metaheuristic instances and at the end chooses the best solution among the several parallel runs. Alternatively, domain decomposition parallel metaheuristics proceed by first decomposing the domain of a problem before optimizing different sub-parts of the problem in parallel. This helps in reducing the overall computation time beside maintaining the metaheuristic robustness. As per our knowledge, we are the first to introduce the domain-decomposition parallel LNS for the multi-job employee scheduling problem. Furthermore, the presented LNS repair operator solves a minimized MJ-ESP MILP, which produces the best possible solution for the destroyed sub-problem. This transforms the metaheuristic into a matheuristic which benefits from the power of both the mathematical programming solution methods and the heuristics. Our PLNS outperform the commercial tool WFC, by accelerating the computational time by an average of 80.7% and enhancing the solution quality by an average of 1.7%.

From the results reported for the three presented algorithms, we believe they are an enriching addition to employee scheduling in the industry. Exploiting the parallelization in each of the algorithm steps makes them all suitable for nowadays computational trend.

CHAPTER 8 CONCLUSION AND RECOMMENDATION

We have presented three algorithms for solving employee scheduling problems. The first two deal with the multi-department employee scheduling problem with inter-department transfers. The second is a continuation of the first, where the proposed heuristic computational time is further enhanced. Finally we attack the multi-job employee scheduling problem. In this chapter we revise the three developed heuristics along with their limitations, and we conclude with proposed future research.

8.1 Summary of works

The first developed heuristic is the MP-DH, a multi-phase decomposition heuristic for the employee scheduling problem with inter-department transfers. The first phase solves a mono-department anonymous employee scheduling problem for each department. The solved MILP size is constant for all datasets guaranteeing quick computation for large instances. The first phase gives a good idea about the possible requirement covered only with the department internal employees. Any uncovered time periods are considered as *critical intervals*, i.e., need transferred employees to fulfill them. The second phase solves a one-day anonymous multi-department employee scheduling problem with derived inter-department transfers, for each of the problem horizon days. The word *derived* denotes that not all possible transfers are considered but only the critical interval transfers are. The resulting daily anonymous schedules indicate which department is responsible for fulfilling each of the critical intervals. Using this phase result, we migrate the transfer requirement to the department possessing the to-be-transferred employees. At the end of this phase, every department has a processed internal requirement and a list of transfer requirements, one for each department. The third phase solves the final mono-department employee scheduling problem with derived inter-department transfers. All department schedules are then merged to form the final schedule.

All phases exploit the parallelism possibility between its different MILPs, the first and third phases optimize all departments in parallel, and the second phase optimizes every day-problem in parallel. MP-DH was further tuned to solve the exact problem presented by Dahmen et al. [22]. The experimental results show that MP-DH outperforms the heuristics in Dahmen et al. [22] with respect to the computational time, and for large datasets with respect to the solution quality as well.

Next, the MP-DH computational time is further reduced by up to 87% on average for large instances, using *HH*, a hybrid heuristic. *HH* uses two models interchangeably for the third phase of MP-DH. The first model is the basic model already presented in MP-DH. The second model is a semi-anonymous model for the same problem, where only the schedules of a subset of the department employees are of interest. *HH* can then solve a sequence of such models to yield a complete schedule. *HH* chooses when the basic model or the semi-anonymous model is used. Every department employee schedule optimization process starts by applying the basic model. When the computational time exceeds a given time threshold and the MILP optimality gap is greater than a given value, the solution of the basic model is stopped and a semi-anonymous model is executed. *HH* maintains high solution quality, with a cost gap of only 3.1% with respect to the MP-DH.

In the last part of the thesis we present a parallel LNS metaheuristic for the multi-job employee scheduling problem. The LNS destroy operator chooses, from the current solution, sub-scopes yielding high cost. High cost is caused by over-coverage, open-shift usage, or the employee minimum working hours violation. When a sub-scope is destroyed, any shift belonging to such sub-scope is deleted. The repair operator then re-constructs the destroyed solution. We use the multi-job employee scheduling problem MILP of the WFC tool as the repair operator for the destroyed parts. The parallel version of the destroy operator creates several disjoint sub-scopes to be destroyed: each is destroyed in a separate parallel thread and repaired with a standalone version of the repair operator. After all threads have repaired their own destroyed part of the solution, a new solution is created by merging the repaired part of each thread. The parallel LNS algorithm succeeded in improving both the computational time and the solution quality with respect to the WFC tool solution.

8.2 Limitations

The ESP-IDT studied in Chapters 4 and 5 does not consider some important practical features such as the travel times between the departments during employee transfers, employee preferences and the positioning of breaks within the shifts. However, our experimental tests have shown that MP-DH is highly flexible as it was easily transformed to respect the problem setting of Dahmen et al. [22]. We believe that considering a new problem setting can be easily adapted to the heuristic. Furthermore, even if we have developed and tested a variety of ways to expect transfer needs at the first phase, the explicit enumeration of all possible transfers in MP-DH-noP1 outperforms the MP-DH on average. This strategy was, however, deemed unfavorable due to its high time consumption. Nevertheless, it shows that better

transfer requirements can be identified.

One limitation faced by the *hybrid heuristic* is the existence of several parameters: t_l, *emp-Percent* and λ. The values of these parameters may significantly affect the final solution quality and the computational time. Thus good calibration is needed when new problem settings are used.

After destroying a solution in the LNS matheuristic for the multi-job employee scheduling problem, the repair operator re-optimizes the full problem, while maintaining the fixed shift (S_{fixed}) variables to one in order to preserve these shifts. Faster execution could be achieved by only enumerating the needed shifts, those belonging to the destroyed sub-scopes, and adjusting the problem constraints. In our research scope, this was not easy to implement in the WFC solver.

8.3 Future research

The MP-DH first phase is responsible for critical intervals extraction. An equivalent procedure is the explicit enumeration of all possible critical intervals, like in MP-DH-noP1, which induces large computational times in the second phase, because all transfer shifts are enumerated from all departments to each critical interval. Enhancing the critical intervals extraction without reaching an excessive enumeration is an interesting future research objective.

For the second phase of MP-DH, local search can be tested to replace the daily anonymous scheduling MILP. Eventually, local search can be further used to enhance the MP-DH final solution by adjusting or adding transfer shifts.

Regarding the MJ-ESP parallel LNS heuristic, several of its components can be investigated in future works. First, collaborative parallel local search can be exploited, which shows good success in the literature. Where several parallel threads optimize the problem and share the features of the good computed solutions or the solutions themselves among each other. Second, the adaptive LNS heuristic, where several neighborhoods (destroy/repair operators) are used in an adaptive way. Adaptive LNS updates a score for each destroy/repair operator after each iteration, depending on the level of achieved improvement. At each iteration, a destroy/repair operator is chosen depending on its score. Thus sufficient number of iterations is needed in order to frequently update these scores. In our LNS, we had

small number of iterations due to the repair operator long computational time. Reduction in the computational time can be done by reducing the destroy percentage for the first set of iterations of the metaheuristic, then increasing it when LNS stops improving the current solution. In our tests, using a small destroy percentage is equivalent to a large one during the early iterations, but could not converge at the end of the metaheuristic compared to the large destroy percentage. An adaptive LNS strategy may avoid this drawback.

REFERENCES

[1] U. Aickelin, E. K. Burke, and J. Li, "An evolutionary squeaky wheel optimization approach to personnel scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 433–443, 2009.

[2] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.

[3] T. Aykin, "Optimal shift scheduling with multiple break windows," *Management Science*, vol. 42, no. 4, pp. 591–602, 1996.

[4] ——, "A comparative evaluation of modeling approaches to the labor shift scheduling problem," *European Journal of Operational Research*, vol. 125, no. 2, pp. 381–397, 2000.

[5] K. R. Baker, "Workforce allocation in cyclical scheduling problems: A survey," *Operational Research Quarterly*, pp. 155–167, 1976.

[6] K. R. Baker and M. J. Magazine, "Workforce scheduling with cyclic demands and day-off constraints," *Management Science*, vol. 24, no. 2, pp. 161–167, 1977.

[7] J. F. Bard and H. W. Purnomo, "Hospital-wide reactive scheduling of nurses with preference considerations," *IIE Transactions*, vol. 37, no. 7, pp. 589–608, 2005.

[8] J. F. Bard and L. Wan, "The task assignment problem for unrestricted movement between workstation groups," *Journal of Scheduling*, vol. 9, no. 4, pp. 315–341, 2006.

[9] ——, "Workforce design with movement restrictions between workstation groups," *Manufacturing & Service Operations Management*, vol. 10, no. 1, pp. 24–42, 2008.

[10] S. E. Bechtold and L. W. Jacobs, "Implicit modeling of flexible break assignments in optimal shift scheduling," *Management Science*, vol. 36, no. 11, pp. 1339–1351, 1990.

[11] O. Berman, R. C. Larson, and E. Pinker, "Scheduling workforce and workflow in a high volume factory," *Management Science*, vol. 43, no. 2, pp. 158–172, 1997.

[12] R. Bürgy, H. Michon-Lacaze, and G. Desaulniers, "Employee scheduling with short demand perturbations and extensible shifts," *Omega*, vol. 89, pp. 177–192, 2019.

[13] E. K. Burke *et al.*, "A scatter search methodology for the nurse rostering problem," *Journal of the Operational Research Society*, pp. 1667–1679, 2010.

[14] C. Canon, "Personnel scheduling in the call center industry," Ph.D. dissertation, Université François-Rabelais de Tours, 2007.

[15] J.-F. Cordeau *et al.*, "Scheduling technicians and tasks in a telecommunications company," *Journal of Scheduling*, vol. 13, no. 4, pp. 393–409, 2010.

[16] M.-C. Côté, B. Gendron, and L.-M. Rousseau, "Grammar-based column generation for personalized multi-activity shift scheduling," *INFORMS Journal on Computing*, vol. 25, no. 3, pp. 461–474, 2013.

[17] T. G. Crainic, *Parallel Metaheuristic Search*. Springer, 2018.

[18] T. G. Crainic and N. Hail, "Parallel metaheuristics applications," *Parallel Metaheuristics: A New Class of Algorithms*, pp. 447–494, 2005.

[19] T. G. Crainic and M. Toulouse, "Parallel meta-heuristics," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Springer, 2010, pp. 497–541.

[20] S. Dahmen and M. Rekik, "Solving multi-activity multi-day shift scheduling problems with a hybrid heuristic," *Journal of Scheduling*, vol. 18, no. 2, pp. 207–223, 2015.

[21] S. Dahmen, M. Rekik, and F. Soumis, "An implicit model for multi-activity shift scheduling problems," *Journal of Scheduling*, vol. 21, no. 3, pp. 285–304, 2018.

[22] S. Dahmen *et al.*, "A two-stage solution approach for personalized multi-department multi-day shift scheduling," *European Journal of Operational Research*, vol. 280, no. 3, pp. 1051–1063, 2020.

[23] G. B. Dantzig, "Letter to the editor-a comment on edie's "traffic delays at toll booths"," *Journal of the Operations Research Society of America*, vol. 2, no. 3, pp. 339–341, 1954.

[24] P. De Bruecker *et al.*, "Workforce planning incorporating skills: State of the art," *European Journal of Operational Research*, vol. 243, no. 1, pp. 1–16, 2015.

[25] M. Defraeye and I. Van Nieuwenhuyse, "Staffing and scheduling under nonstationary demand for service: A literature review," *Omega*, vol. 58, pp. 4–25, 2016.

[26] F. Della Croce and F. Salassa, "A variable neighborhood search based matheuristic for nurse rostering problems," *Annals of Operations Research*, vol. 218, no. 1, pp. 185–199, 2014.

[27] S. Demassey, G. Pesant, and L.-M. Rousseau, "Constraint programming based column generation for employee timetabling," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, R. Barták and M. Milano, Eds.

[28] E. Demirović and N. Musliu, "Maxsat-based large neighborhood search for high school timetabling," *Computers & Operations Research*, vol. 78, pp. 172–180, 2017.

[29] G. Desaulniers *et al.*, "Crew scheduling in air transportation," in *Fleet Management and Logistics*, T. G. Crainic and G. Laporte, Eds. Springer, 1998, ch. 8, pp. 169–185.

[30] G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, no. 1, pp. 161–175, 1990.

[31] F. F. Easton and D. F. Rossin, "A stochastic goal program for employee scheduling," *Decision Sciences*, vol. 27, no. 3, pp. 541–568, 1996.

[32] L. C. Edie, "Traffic delays at toll booths," *Journal of the Operations Research Society of America*, vol. 2, no. 2, pp. 107–138, 1954.

[33] A. T. Ernst *et al.*, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*, vol. 153, no. 1, pp. 3–27, 2004.

[34] B. Faaland and T. Schmitt, "Cost-based scheduling of workers and equipment in a fabrication and assembly shop," *Operations Research*, vol. 41, no. 2, pp. 253–268, 1993.

[35] C.-N. Fiechter, "A parallel tabu search algorithm for large traveling salesman problems," *Discrete Applied Mathematics*, vol. 51, no. 3, pp. 243 – 267, 1994.

[36] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[37] D. Godard, P. Laborie, and W. Nuijten, "Randomized large neighborhood search for cumulative scheduling." in *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, vol. 5, 2005, pp. 81–89.

[38] M. Hojati and A. S. Patil, "An integer linear programming-based heuristic for scheduling heterogeneous, part-time service employees," *European Journal of Operational Research*, vol. 209, no. 1, pp. 37–50, 2011.

[39] D. Hur, V. A. Mabert, and K. M. Bretthauer, "Real-time work schedule adjustment decisions: An investigation and evaluation." *Production and Operations Management*, vol. 13, no. 4, pp. 322–339, 2004.

[40] J. Jin, T. G. Crainic, and A. Løkketangen, "A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems," *European Journal of Operational Research*, vol. 222, no. 3, pp. 441 – 451, 2012.

[41] J. Jin, "Pré-affectation des tâches aux employés effectuant des tâches non-interruptibles et des activités interruptibles," Ph.D. dissertation, École Polytechnique de Montréal, 2009.

[42] Ö. Kabak *et al.*, "Efficient shift scheduling in the retail sector through two-stage optimization," *European Journal of Operational Research*, vol. 184, no. 1, pp. 76–90, 2008.

[43] A. Kasirzadeh, M. Saddoune, and F. Soumis, "Airline crew scheduling: Models, algorithms, and data sets," *EURO Journal on Transportation and Logistics*, vol. 6, no. 2, pp. 111–137, 2017.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[45] A. A. Kovacs *et al.*, "Adaptive large neighborhood search for service technician routing and scheduling problems," *Journal of Scheduling*, vol. 15, no. 5, pp. 579–600, 2012.

[46] S. A. Kravitz and R. A. Rutenbar, "Placement by simulated annealing on a multiprocessor," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 6, no. 4, pp. 534–549, 1987.

[47] P. Laborie and D. Godard, "Self-adapting large neighborhood search: Application to single-mode scheduling problems," *Proceedings of the 3rd Multi-disciplinary International Scheduling Conference:Theory and Applications.*, vol. 8, 2007.

[48] N. Lahrichi *et al.*, "An integrative cooperative search framework for multi-decision-attribute combinatorial optimization: Application to the m d p v r p," *European Journal of Operational Research*, vol. 246, no. 2, pp. 400–412, 2015.

[49] A. Legrain, H. Bouarab, and N. Lahrichi, "The nurse scheduling problem in real-life," *Journal of Medical Systems*, vol. 39, no. 1, p. 160, 2015.

[50] Q. Lequy *et al.*, "Assigning multiple activities to work shifts," *Journal of Scheduling*, vol. 15, no. 2, pp. 239–251, 2012.

[51] J. S. Loucks and F. R. Jacobs, "Tour scheduling and task assignment of a heterogeneous work force: A heuristic approach," *Decision Sciences*, vol. 22, no. 4, pp. 719–738, 1991.

[52] R. R. Love Jr. and J. M. Hoey, "Management science improves fast-food operations," *Interfaces*, vol. 20, no. 2, pp. 21–29, 1990.

[53] S. Martin *et al.*, "Cooperative search for fair nurse rosters," *Expert Systems with Applications*, vol. 40, no. 16, pp. 6674–6683, 2013.

[54] H. Michon-Lacaze, "Élaboration de quarts de travail robustes aux perturbations de courtes durée," Master's thesis, École Polytechnique de Montréal, 2016.

[55] H. H. Millar and M. Kiragu, "Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming," *European Journal of Operational Research*, vol. 104, no. 3, pp. 582–592, 1998.

[56] L. F. Muller, "An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem," in *MIC 2009: The VIII Metaheuristics International Conference*, 2009.

[57] E. Munezero, "Une heuristique en deux phases pour la confection d'horaires de personnel avec transferts inter-départementaux d'employés," Master's thesis, École Polytechnique de Montréal, 2014.

[58] L.-M. Munguía *et al.*, "Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs," *Computational Optimization and Applications*, vol. 69, no. 1, pp. 1–24, Jan 2018.

[59] L. Perron, P. Shaw, and I. Sa, "Parallel large neighborhood search," 2003.

[60] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & Operations Research*, vol. 34, no. 8, pp. 2403–2435, 2007.

[61] ——, "Large neighborhood search," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Springer, 2010, pp. 399–419.

[62] C.-G. Quimper and L.-M. Rousseau, "A large neighbourhood search approach to the multi-activity shift scheduling problem," *Journal of Heuristics*, vol. 16, no. 3, pp. 373–392, 2010.

[63] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.

[64] ——, "A unified heuristic for a large class of vehicle routing problems with backhauls," *European Journal of Operational Research*, vol. 171, no. 3, pp. 750–775, 2006.

[65] M. Sabar, B. Montreuil, and J.-M. Frayret, "Competency and preference based personnel scheduling in large assembly lines," *International Journal of Computer Integrated Manufacturing*, vol. 21, no. 4, pp. 468–479, 2008.

[66] D. Sacramento, D. Pisinger, and S. Ropke, "An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones," *Transportation Research Part C: Emerging Technologies*, vol. 102, pp. 289–315, 2019.

[67] L. Saviniec, M. O. Santos, and A. M. Costa, "Parallel local search algorithms for high school timetabling problems," *European Journal of Operational Research*, vol. 265, no. 1, pp. 81–98, 2018.

[68] G. Schrimpf *et al.*, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[69] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 1998, pp. 417–431.

[70] S. Souissi, "Ré-optimisation d'horaires de personnel en ajoutant des transferts entre départements," Master's thesis, École Polytechnique de Montréal, 2016.

[71] L. Talarico and P. A. M. Duque, "An optimization algorithm for the workforce management in a retail chain," *Computers & Industrial Engineering*, vol. 82, pp. 65–77, 2015.

[72] G. M. Thompson, "Labor scheduling: A commentary," *Cornell Hotel and Restaurant Administration Quarterly*, vol. 44, no. 5-6, pp. 149–155, 2003.

[73] J. Van den Bergh *et al.*, "Personnel scheduling: A literature review." *European Journal of Operational Research*, vol. 226, no. 3, pp. 367–385, 2013.

[74] A. Volgenant, "A note on the assignment problem with seniority and job priority constraints," *European Journal of Operational Research*, vol. 154, no. 1, pp. 330–335, 2004.

[75] M. Wen *et al.*, "An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem," *Computers & Operations Research*, vol. 76, pp. 73–83, 2016.

[76] P. D. Wright and S. Mahar, "Centralized nurse scheduling to simultaneously improve schedule cost and nurse satisfaction," *Omega*, vol. 41, no. 6, pp. 1042–1052, 2013.