Titre: Title:	A Divide-and-Conquer Approach to Employee Scheduling
Auteur: Author:	Nicolas Heutte
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Heutte, N. (2020). A Divide-and-Conquer Approach to Employee Scheduling [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/5362/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A divide-and-conquer approach to employee scheduling

NICOLAS HEUTTE
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A divide-and-conquer approach to employee scheduling

présenté par **Nicolas HEUTTE**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Gilles PESANT, président
Daniel ALOISE, membre et directeur de recherche
Guy DESAULNIERS, membre et codirecteur de recherche
Claudio CONTARDO, membre

DEDICATION

To my fellow students,

May our paths cross again.

ACKNOWLEDGEMENTS

I'd like to thank everyone who supported me during my studies at Polytechnique Montréal.

I am grateful to both of my supervisors, Daniel Aloise and Guy Desaulniers, who supported and helped me during this program.

I thank KRONOS for the financial support and the experimental data they provided me during this master.

I'd also like to thank François Lessard for his help during this project. His expertise of KRONOS's software proved invaluable.

I am also gratefully indebted to Rachid Hassani, who, in this period of social distancing, kindly ran experiments for me despite having no obligation to do so.

I give thanks to Gilles Pesant and Claudio Contardo, who accepted to be part of the jury evaluating my work.

Last but not least, I am grateful to the friends and acquaintances I made in this country, for they made studying away from home more pleasant.

RÉSUMÉ

De nos jours, les grandes entreprises ont trop d'employés pour se permettre de déterminer leurs emplois du temps à la main. Par conséquent, ces entreprises sont contraintes d'utiliser des méthodes informatiques pour créer un horaire satisfaisant. De nombreux algorithmes ont été proposés pour générer les meilleurs horaires possibles. Cependant, leur complexité temporelle rend la résolution de gros problèmes de génération d'horaire très lente. On cherche donc un moyen de réduire cette complexité lorsque l'on est confronté à des problèmes de grande taille.

La piste qui sera explorée dans ce mémoire est l'application du principe de "diviser pour régner" au problème de génération d'horaire. Si l'on parvient à diviser un exemplaire de grande taille en plusieurs sous-exemplaires plus petits, on peut résoudre chaque sous-exemplaire séparément et déduire des solutions obtenues une solution à l'exemplaire de grande taille que l'on souhaitait traiter en premier lieu. Cependant, la structure du problème de génération d'horaire fait que la manière de séparer l'exemplaire de grande taille en sous-exemlaires peut très fortement dégrader la qualité des solutions obtenues. On souhaite donc trouver un moyen de partitionner un exemplaire de grande taille sans que la qualité de la solution obtenue ne diminue trop.

Pour cela, deux approches sont proposées dans ce mémoire. Tout d'abord, la méthode dite séquentielle est présentée. Elle consiste à déterminer heuristiquement des caractéristiques de haut niveau de l'exemplaire traité, puis on se sert de ces caractéristiques pour calculer quels jobs et quels employés devraient appartenir au même sous-exemplaire. On exécute ensuite un clustering pour obtenir des sous-exemplaires qui respectent effectivement ce critère. Quand à elle, la méthode dite holistique se base sur une vision d'ensemble du problème. En effet, on indroduit d'abord une simplification du problème de conception d'horaire, puis on résout ladite simplification. Ensuite, on se sert des résultats obtenus pour déduire une partition adéquate de l'exemplaire de grande taille.

Pour finir, on compare les résultats obtenus avec ces deux méthodes aux résultats d'un logiciel commercial de génération d'horaire. Dans la plupart des cas, nos algorithmes produisent des horaires de meilleure qualité, mais ils prennent plus de temps à être déterminés.

ABSTRACT

As companies grow larger and larger, it becomes necessary to determine the work schedules of a large number of employees. Moreover, laws and labor agreements grow more complex over time, which in turn makes the problem of determining a schedule a difficult one. Of course, many computational methods to solve personnel scheduling problems have been proposed, but most of them have a time complexity which makes them unsuitable for instances that are particularly large. Therefore, there is a demand for an algorithm that is able to quickly handle large instances of schedule generation problems.

Since plenty of progress have already been made in making the solving algorithms better, we will attempt to make the problems smaller to decrease the computation times. In other words, we will apply the divide-and-conquer principle to this issue. If we could split a large personnel scheduling instance into multiple subinstances, we could apply any state-of-the-art method of our choosing to the subinstances to compute a solution to the master instance. Since the subinstance's sizes are a fraction of the original instance's, we could expect a sizable reduction of the computation time. On the other hand, if the original instance is split in an inadequate manner, the quality of the final results could be disastrous.

In this master's thesis, we propose two families of methods to split personnel scheduling instances. First are the sequential methods, where use a sequence of heuristics to estimate which jobs and employees should be together, and then we use clustering to obtain our final partition. Second are the holistic methods, where we introduce a related and simpler problem, solve it and then deduce a good partition from the solution to that problem.

We then compare the results obtained using these methods to those produced by a commercial personnel scheduling software. In most cases, using our instance partitioning algorithms allows us to obtain better solutions, at the cost of a longer computation time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

VNS	Variable Neighbourhood Search
MIP	Mixed Integer Program/Mixed Integer Programming
LP	Linear Program/Linear Programming

CHAPTER 1 INTRODUCTION

Employee wages are a significant part of a company's expenses. Therefore, it is in its best interest to ensure that the employees' work is as efficient as possible. One possible strategy to work towards that goal is to attempt to create employee schedules that cost as little as possible, matching the projected demand for the employees' work and the employees' personal preferences. For a long time, creating such a schedule was a task left to a human employee, but such methods are no longer feasible due to a number of reasons. First, companies grow bigger. While it is possible to create a schedule for a few employees "by hand", it is quite unrealistic to do so for a few dozens of employees or more. Another reason why a large schedule cannot be determined by hand is that the number of constraints to take into account due to laws and labor agreements grows regularly, which makes creating even a small legal schedule a difficult task. Because of these reasons, quite a few large companies have turned to computer science to generate their employees' schedules, for a computer running a suitable algorithm can generate better, more equitable schedules in a fraction of the time it would take to a human to do the same.

1.1 Definitions and basic concepts

There is no set standard definition of a personnel scheduling problem, though many share similar characteristics. Ideally, we would like to be able to use our work on most personnel scheduling problems, so we will present a general definition of a personnel scheduling problem which contains only the basic information contained in personnel scheduling problems. This definition should be flexible enough to fit many actual personnel scheduling problems. The algorithms presented within this thesis will thus be applicable to all problems fitting this loose definition.

An instance of a personnel scheduling problem usually consists of a planning horizon, a set of jobs, a set of employees, some rules and a cost function:

- The planning horizon is the period on which the user wants to generate the schedule. It is generally at least a week long. Usually, this horizon is discretized into periods, whose length depends on the precision the user needs. This thesis will focus on personnel scheduling problems from the retail industry, where the periods usually last between one minute and one hour. In some other industries, such as healthcare, as few as three periods per day can be used.

- The set of jobs contains all the jobs that we want to be worked by the employees. Each job has a demand curve that indicates how many employees the company wants to be working that job, for each period in the planning horizon. The figure 1.1 shows a possible demand curve. In this case the planning horizon is 24 hours long, with a period length of one hour. The job that has this demand curve requires one employee to be working from 5 am to 8 am, two from 8 am to 6 pm and one from 6 pm to 9 pm.

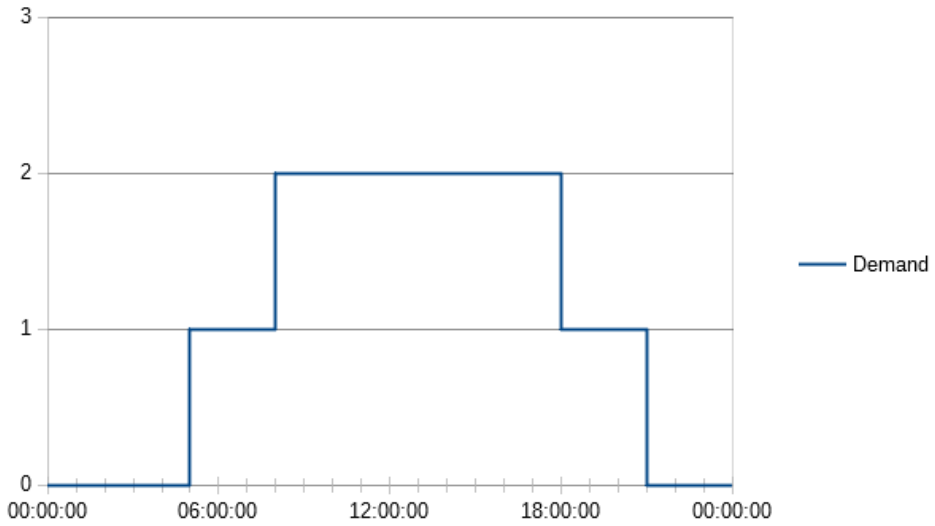


Figure 1.1 Example of a demand curve

- The set of employees contains all the employees that are available to work the jobs. Data on each employee includes when and which jobs they can work. The employee's preference regarding which job they want to work and when they want a day off are also occasionally available.

- The rules are constraints that make sure that any solution to the personnel scheduling problem respects the local law, the labor agreements of the company, as well as any additional rule the company decides to add. Common rules involve limiting the total number of shifts an employee can work over the planning horizon, requiring that an employee works at only one job for each of its shifts, constraining the length of the shifts so they are neither too short nor too long or limiting the number of hours an employee can work over a week.

- Lastly, the cost function defines what is considered to be a "good" solution. For every solution that conforms to the rules, it gives us the cost of that solution. Solving an instance of a personnel scheduling problem means finding a solution that satisfies the

rules at a minimal cost. As finding the optimal solution is often too time-consuming, we are usually satisfied with finding one whose cost is merely low enough. Almost all cost functions penalize having a different number of employees than what the demand curves requires. When there is not enough employees, we are in a situation called *under-coverage*, which is harshly penalized. When there are too many, we are in *over-coverage*, which is more lightly penalized.

Occasionally, some situations, such as having employees work overtime or be out of work, are seen as undesirable but tolerable if they are necessary. This is modeled in a personnel scheduling problem by adding an additional cost penalty if that undesirable situation is present.

1.2 Motivation

The astute reader may notice that under the previous assumptions, a personnel scheduling instance, which we will call the *master instance* hereafter, can be split in several sub-instances. Indeed, if we split the employee set and the job set, as well as duplicate the planning horizon, rules and cost function, we obtain new instances of the personnel scheduling problem. It is not given that the rules and cost function stay adapted to these new instances, but in practice most of the rule sets and cost functions used focus on one job or employee at a time. Thus, for an arbitrary partition of the employee and the job set, splitting the rules appropriately is not an issue.

As we can separate a master instance into as many smaller instances as we desire, one automatically ponders whether the divide-and-conquer technique can be used. If we use an arbitrary partition of the master instance, each sub-instance can be solved by any algorithm of our choice. Since these solutions are sets of shifts, it is trivial to compute their union to get a solution to the master instance. The master solution we obtain this way is feasible in the sense that it respects the rules of the master problem. However, nothing guarantees us that its cost is even remotely close to the optimal cost. For example, if a subinstance contains one job and no employees qualified for that job, it would be severely *under-covered*. Therefore, if we want the cost of the master solution to be as low as possible, the partition we use must be chosen with great care.

To be more precise, by partitioning the master instance, we are adding the constraint that an employee may only work at jobs that belong to the same element of the partition. Ideally, the partition would be chosen in such a way that this constraint has no impact on the cost of the optimal solution to the master instance. However, the existence of a nontrivial partition

meeting that constraint is not guaranteed. Therefore, minimizing the increase in solution cost caused by the partitioning of the instance will be an important criteria in choosing the partition.

To minimize the increase of the solution cost, minimizing the impact of the partitioning constraint seems to be a sound strategy. Ideally, we would want that for the partition used, there exists some optimal solution to the master instance such that in this solution, each employee works at jobs that belong to the same subinstance. That way, combining the optimal solutions of the subinstances will result in an optimal solution to the master instance.

Since the main reason we want to apply the divide-and-conquer principle to this problem is to reduce the time needed to obtain a solution, the other main criterion for choosing a partition will be how much computation time we save thanks to it. In practice, the best improvements are expected when the subinstances are of similar sizes.

To sum up, we want to create an algorithm that produces a partition that is both balanced and of high-quality. The former criterion ensures a reduction of the computation time, while the latter limits the loss of quality of the obtained solution to the master instance.

We will need some way to judge the quality of our results. Thanks to our partnership with the company KRONOS, we have access to their schedule generation software as a reference point, as well as to several datasets to test our algorithms on.

Some algorithms, such as the one used by KRONOS's software, make a simple and suboptimal partition in order to quickly obtain a feasible solution. Since that solution is most likely not optimal, algorithms that aim to improve a solution's quality are then used. Hopefully that will result in a solution whose cost is nearly optimal.

1.3 Research objectives

In this master's thesis, we will explore the possibility of making a high-quality partition of the master instance. KRONOS's reference algorithm already uses a simple partitioning method, which we hope to outperform in terms of solution cost. However, since the proposed algorithms are likely to be more complex, it seems unlikely that the necessary computation time will be significantly lowered. Ideally, we would like to have results that are better than KRONOS's reference's, while keeping the computation time reasonable.

Since partitioning instances is more difficult and efficient the larger the instance is, we expect to see the limited improvements over KRONOS's reference on the smaller instances, while the results on larger ones will be better.

1.4 Thesis structure

We will first perform a literature review on two different topics. First, we will present previous works dealing with the personnel scheduling problem. Second, a few clustering methods will be presented in order to provide a background for the clustering methods we will use.

The algorithms we devised, while modular to some extent, fall into two broad categories: the sequential methods and the holistic ones. The former refers to algorithms that build a sequence of heuristics of increasing level. The high-level heuristics so obtained are then used to generate an affinity, which is itself used in a clustering algorithm, giving us the partition we want. We shall present these algorithms and their results in Chapter 3.

The holistic methods are a completely different approach to the problem. In this case we introduce a simplified version of the personnel scheduling problem, which we combine with the constraint of expliciting a partition. The resulting simpler instance is then solved using a variation of a gradient descent algorithm. The clustering which is obtained by solving the simplified problem should then be appropriate for the master instance. These algorithms, as well as their results, will be presented in Chapter 4.

Lastly, we shall summarize our findings, point out the weaknesses of our work, as well as outline several avenues for future research.

CHAPTER 2 LITERATURE REVIEW

In this chapter, we will present a literature review regarding two domains. First, a general overview of personnel scheduling related works will be presented, with a focus on LP-based methods. Second, since this thesis will use several clustering algorithms, a few notions of cluster analysis will be presented.

2.1 Personnel scheduling

2.1.1 Set covering formulation

Personnel scheduling problems have been studied for more than sixty years. In 1954, Edie studied the New York Port Authority toll collection [1]. He was mainly concerned with properly modeling the expected demand for toll collectors than solving the actual scheduling problem, though the need for a systematic way to solve the personnel scheduling problem he introduced was highlighted.

Later in the same year, the first solution to Edie's problem was proposed by Dantzig [2]. He created a linear program modeling the scheduling problem, allowing it to be solved using the simplex algorithm [3]. It mainly consisted of a decision variable for the number of employees starting work at a given time following a given *shift pattern*, which is a sequence of rest periods and work periods that can legally make up a shift. It is then easy to add constraints making sure there are more employees working than the expected demand at a given time period. It is also trivial to implement the objective of minimizing the total number of shifts needed.

While Dantzig's method may have been adequate for the Port Authority's situation, it is quite difficult to extend to more complex instances of personnel scheduling. Indeed, the number of variables in Dantzig's linear program is proportional to the number of possible shifts an employee can work. In the case of the NY Port Authority, the only variable characteristic of a shift was the position of two breaks in one's shift. The corresponding increase in the number of variables was thus manageable. But if one adds variable shift length, more than one job and/or a variable break time, the number of variables skyrockets, making the computation time needed to obtain a solution too high.

2.1.2 Implicit models

A way to reduce the size of the LP models used was necessary. In 1990, Bechtold and Jacobs proposed an alternative to Dantzig's formulation [4]. Their approach was to avoid enumerating all shift patterns explicitly. To do so, instead of using one variable per employee to decide which shift pattern is selected, two are used: one represents the beginning and the end of the shift, and the other represents when the break is taken during the shift. That way, the number of variables decreased from roughly (number of possible shift types * number of possible break positions) to approximately (number of possible shift types + number of possible break positions). It is therefore much more efficient than Dantzig's formulation to handle scheduling problems where employees work shifts containing a flexible break period.

Aykin [5] expands Bechtold's formulation to handle shifts with multiple flexible breaks, each within disjoints time intervals. Instead of enumerating every possible shift, he merely enumerated the shifts that did not take breaks into account, and then added decision variables representing when the breaks are taken in a given shift. That method trades an increase in the number of constraints in the LP model for a significant reduction of the number of variables.

Bechtold's formulation relies on the hypothesis that there is no *extraordinary overlap*, which means that no break is strictly included in another break. Addou and Soumis [6] managed to generalize Bechtold's formulation so that it does not require that hypothesis. Additionally, having extraordinary overlap may lead to a slightly lower number of constraints compared to Bechtold's original formulation.

While this improves the computation time required to solve the LP model, it merely delays the inevitable. Indeed, any attempt to generalize that formulation, such as having more than one job to fill or allowing a shift to have variable length, will cause the number of decision variables to skyrocket again.

2.1.3 Formal language-based methods

Côté et al. proposed an interesting alternative to the shift enumeration required by the previously mentioned methods. In [7], the shift patterns are never explicited directly, but the employees's rest and work periods are constrained by a set of linear constraints generated from the actual rules that a valid shift pattern must match. Any set of rules that can be expressed through an automata [8] that accepts valid shift patterns and rejects illegal ones can be used by this method. Overall, this approach offers competitive results with the rest of the literature.

2.1.4 Multi-jobs shifts

The set covering and the implicit formulation deal with a single job. They can be extended to work in the case where there are several jobs to be worked. However, when the employees are allowed to switch jobs during a shift, the number of shift patterns to consider becomes truly enormous. Therefore, there is a need for a more specialized method to tackle this case.

Attia et al. [9] designed a solution that works in a particular context: she assumed that each employee has a fixed main job, and that working other jobs should be exceptional. Her proposal consists in attempting to cover each job's demand using employees that have that job as a main job, using regular mono-job methods. Then, an inter-job demand is computed for each pair of jobs using the over and under-coverages obtained. Lastly, the mono-jobs problems from the first step are solved while taking the inter-job demand into account. This method produces good results while requiring a reasonable computation time.

2.1.5 Employee preferences

Having a schedule that fits the employees's personal preferences is seldom incompatible with having an efficient schedule. Instead of using the antiquated method of finding a fellow employee to swap shifts with, ways to take into account each employee's personal desires into account at the schedule generation stage have been studied.

Mohan [10] studied a simple way to take employee preferences into account. While the proposed LP model does indeed have an objective function that maximizes the employees satisfaction weighted by seniority, the model itself is based off Dantzig's, which makes it quite slow to be solved as soon as there are many different shift patterns. Undeterred by this, he also studied the effect of adding cuts to the branch-and-bound usually used to solve MIPs. Adding cuts significantly improved the computation time on a majority of instances, though that may be specific to the context of his study, which focuses on workforce mostly made of part-time employees.

Stolletz and Brunner [11] introduce a more complex formulation to take employee preferences into account. First, a solution to the personnel scheduling problem of minimal cost is found using a LP model. Then, the constraint of having a cost equal to the minimal cost previously found is added. The objective function then changes to maximizing employee preferences. Then, the same process is used again to optimize a notion of fairness while the satisfaction score of the employees is fixed. In this case, the fairness objective used was to equally distribute working hours and on-call periods amongst the employees. Of course, many other definitions of fairness could be used, depending on the characteristics of the problem at

hand. Unfortunately, the complexity of this solution makes it applicable only to relatively small instances.

2.1.6 Dealing with uncertainty

It is important to remember that the data used to construct a personnel scheduling instance is often unreliable. Demand curves for a variety of jobs are often mere educated guesses, which may prove true or erroneous. The employees are also prone to being late or unable to come to work, which can be another source of erroneous information. Therefore, ways to reduce the impact of these errors were examined.

To deal with an inexact demand, Froger [12] proposed a method to re-optimize an existing solution meeting the expected demand. The re-optimized solution should then meet the updated demand. The proposed method consists in choosing which shifts of the original solution stay fixed, and which ones are allowed to be modified. Then, an additional cost for modifying existing shifts is added, since it is undesirable to modify the schedule too much. Lastly, the solution is re-optimized. Since many shifts are now fixed, the size of the linear program is significantly smaller, which means that the computation time needed to update the schedule is very short.

Michon-Lacaze et al. [13] propose a different algorithm dealing only with short-term demand perturbations. It introduces the notion of *potential demand*, which is a possible extra demand. Then, for each shift, the *potential work* is computed. It is the work that an employee could do on top of its scheduled work, if they are called early or asked to stay at work longer. Then, in the MIP used to generate the solution, an extra cost is added to the objective if the potential work is unable to cover the potential demand. Since having a late employee is quite similar to having an extra demand of 1 at the beginning of that employee's shift, this method can also be used to deal with the employees's tardiness.

2.1.7 State of the art

The modern shift-scheduling methods use a variety of tricks to keep computation times reasonable. Ernst et al. [14] and Van den Bergh et al. [15] wrote complete state-of-the-arts referencing many modern methods. Ernst et al. differentiate six general categories of scheduling-related problems to be solved, while Van den Bergh et al. classify existing methods according to various different characteristics.

2.2 Clustering

In this thesis, we will also use a few notions of clustering, so we will present a few basic concepts of this domain. When you have a set of data points, a clustering algorithm will compute groups of data points, called *clusters*, such that the elements of each cluster are close to the elements in the same group and far away from the elements belonging to other clusters. An example of a possible clustering can be seen in figure 2.1.

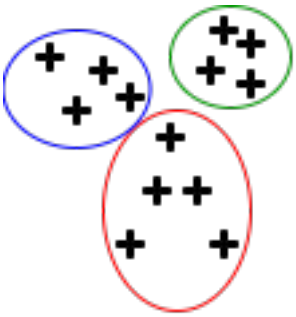


Figure 2.1 Example of a possible clustering

In this example, the points within the green circle are close to each other and relatively farther away from the other points. Do note that the clustering represented in the figure is not the only possible clustering of this set of data points.

Clustering algorithms need a way to know how close two data points are. The most natural metric for numerical data is the Euclidean distance between points of $\mathbb{R}^n$. If data points are not easily represented by a point of $\mathbb{R}^n$, some algorithms can function with an affinity or dissimilarity between all pairs of points.

The most well-known clustering algorithm is perhaps the K-means algorithm [16]. It works only on data that can use the Euclidean distance as a measure of closeness. When it is the case, one arbitrarily chooses K points in $\mathbb{R}^n$ as cluster centers. Then, until the clusters stabilize, one first updates the assignment of a point to a cluster by associating them with the closest cluster center, then one updates the cluster centers to be the average of their cluster's points. This method has the merit of being simple and quick, but it is sensitive to its initialization.

Another family of clustering algorithm is that of the hierarchical clustering methods [16]. It involves constructing a hierarchy of clusters, where the top is a single cluster encompassing all data points. Then, each other level contains a clustering where a cluster from the upper level is split in two. This allow the user to pick a clustering with any number of clusters he wants. There are generally two ways to construct such a hierarchy:

- The divisive approach, where one starts with a cluster containing all data points, and then splits a cluster at each step, which gives us the desired hierarchy.
- The agglomerative approach, where we start with each point being its own cluster. Then, at each step, two clusters are chosen to be merged, until we have only one cluster left.

The agglomerative approach is the one we will use in this thesis, so we will elaborate a bit on how the clusters to be merged are chosen. Let us assume the closeness of data points is measured using a dissimilarity measure. Then, most methods extend that measure in some way to be defined between clusters as well as between points.

If we have two clusters A and B and a dissimilarity d, the most common extensions are as follow:

- Single linkage: $d(A, B) = \min_{a \in A, b \in B} d(a, b)$. It is similar to the distance between sets.
- Complete linkage: $d(A, B) = \max_{a \in A, b \in B} d(a, b)$. This distance only takes into account the greatest dissimilarity between a point of A and a point of B.
- Average linkage: $d(A, B) = \frac{\sum_{a \in A, b \in B} d(a,b)}{|A||B|}$. The previous criterion are a bit too focused on a single value of the dissimilarity. This can be problematic in some cases, so the average linkage is a generic compromise.

Some other criterion exist, but they are less common than these ones.

To sum up, the agglomerative approach is made of the following steps:

- Put all data points in their own cluster.
- While there is more than one cluster, do the following:
  - Using the extended measure, find which pair of clusters are the closest.
  - Merge them.
  - Save the obtained clustering.
- Then, a clustering for each possible number of clusters is obtained.

An important sub-field of clustering is constrained clustering. As it name implies, it deals with finding good clusterings that fit some set of constraints. For example, external data can

imply that a particular pair of points belong in the same cluster. It can be required that the sizes of the clusters should be balanced. Or, it can be required that each cluster must have a minimal size.

A possible approach to these kinds of problems is to formulate a clustering problem as a LP model [17]. Then, the user can add any constraints that can be expressed in the LP environment. Once the user has added the constraints of his choice, solving the LP model gives a solution to the constrained clustering problem.

Other approaches usually focus on a specific set of constraints. For example, if balanced clusters are wanted, Costa et al. [18] propose an algorithm based on the Variable Neighbourhood Search (VNS) [19]. It starts by defining the swap operation on two data points as the action of swapping which clusters they are assigned to. Then, balanced clusters are arbitrarily initialized. Lastly, a VNS is performed using the neighbourhoods consisting of the clusters that can be obtained by using a fixed number of swap operations.

CHAPTER 3 SEQUENTIAL METHOD

In this chapter, the sequential method will be presented. First, an overview of each major step will be presented. Then, after defining what data the sequential method needs to run, the detailed computation process will be explained. Lastly, the performance of the sequential method will be compared to KRONOS's reference algorithm's.

3.1 Overview

As described in the introduction, a balanced and high-quality partition is desired. Since the latter can be achieved by putting specific jobs and employees in the same sub-instances, it is natural to use clustering to do so. Employees and jobs that go well together when put in the same subinstance would be identified, then a clustering algorithm would be used to produce the partition, which would be reasonably balanced and of good quality.

Since we will use a clustering algorithm, we will need to compute some kind of measure between the elements we will cluster together.

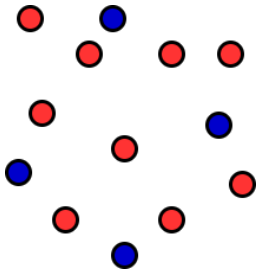
First, let us notice that we need to cluster both the set of employees and the set of jobs. Clustering each separately is impossible, since there is no guarantee that the employees of one cluster will work nicely with the jobs of the corresponding job cluster. It is also inefficient to bluntly cluster the union of the set of employees and of the set of jobs. The reason is slightly subtler: a particular job has a set demand, but regular clustering algorithms do not attempt to put just enough employees in a cluster to cover the cluster's jobs' demand.

Therefore, we settle for the following compromise: we will first compute a mapping from the employee set to the job set, that will represent the assignment of each employee to a job. That way, we can simply cluster the job set and the clustering of the employee set is obtained by computing the sets of employees whose associated job is in a particular cluster. Figure 3.1 gives a visual representation of the process.

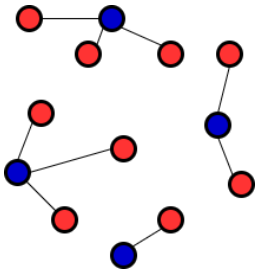
3.2 Available data and notations

Before we proceed, let us explicit what data is available to us. It is assumed that the following data is available, or that it can be deduced from the available data:

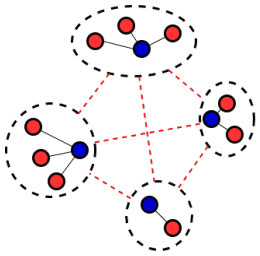
- E is the set of all employees.



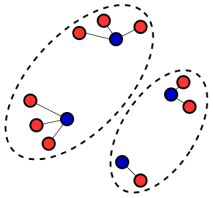
(a) Unclustered data



(b) Employees are assigned to jobs



(c) An affinity is computed between jobs



(d) The clustering gives us the desired partition

(e) Legend

Figure 3.1 Diagram of the clustering process

- J is the set of all jobs.
- $H \subset \mathbb{R}$ is the planning horizon. The actual discretization of that horizon is unimportant in theory, but it might be useful to write an efficient implementation of this algorithm.
- $\forall j \in J, D_j \in (\mathbb{N})^H$ is the demand curve associated with the job $j \in J$. $D_j(t)$ is the number of employees that should bet working at the job $j \in J$ at the time $t \in H$.
- $\forall e \in E, Q_e \subset J$ the set of jobs for which the employee e is qualified. From these sets we also define $Q_j \subset E$ as the set of all employees that are qualified for the job $j \in J$.
- $W_e \in \mathbb{R}$ an estimate of the maximum amount of time the employee e can work during the planning horizon. The value of W_e is not always directly included in an instance's rules, but it can often be deduced from other rules. For example, we could multiply the maximum length of a shift by the maximum number of allowed shifts during the planning horizon.
- $A_e \in \{0,1\}^H$ represent the time during which the employee e is available to work. If $A_e(t) = 1$, the employee $e \in E$ is can work at the time $t \in H$. Otherwise, e is considered unavailable.

That information is present in almost all personnel scheduling instances, though some additional computing might be needed to extract it in the above form.

For example, the qualifications of an employee are sometimes treated as a characteristic of an employee. Each employee has some set of qualifications, such as diplomas, language certifications, and so on. Each job then requires a certain set of qualifications that employees must meet to work at that job. In that case, it is trivial to determine whether a specific employee can work at a specific job, therefore we can compute Q_e and Q_j easily.

In the future, we will need some way to estimate whether some set of employees E_j is likely to be enough to cover some job j's demand. To do that, we introduce to coverage ratio $C(j, E_j)$ as follows:

$$C(j, E_j) = \min(\min_{t \in H} \frac{\sum_{e \in E_j} A_e(t)}{D_j(t)}, \frac{\sum_{e \in E_j} W_e}{\int_H D_j}) \tag{3.1}$$

It is equal to the lowest offer to demand ratio, taken either at each instant of the planning horizon, or over the whole planning horizon. If the coverage ratio is lower than 1, it is impossible to perfectly cover the job's demand. Although it is not true in general, we hope that having this ratio higher than 1 will be sufficient in practice to cover the job's demand.

3.3 Computing employee-job assignment

Our present objective is to compute some function f from E to J such that $\forall j \in J$, $f^{-1}(j)$ can adequately meet the demand D_j. That mapping does not need to be perfect, but the better it is, the better we can expect the results to be.

In order to compute such a mapping, employees will be successively assigned to a job until there are no employee left unassigned. To have a decent final mapping, we will first identify the jobs that we are going to have trouble filling. To do that we will use a heuristic to determine a job's priority. The higher a job's priority, the more difficult it will be to find enough employees to cover the associated demand.

3.3.1 Job priority

Naive definition

At first, we used a very simple heuristic to determine a job's priority. After all, it is trivial to compute both the total number of man-hours a job requires, and the total number of man-hours the employees qualified for that job are able to work. Therefore, the following formula for a job's priority should be decently accurate:

$$P_j = \frac{\int_H D_j}{\sum_{e \in Q_j} W_e} \tag{3.2}$$

When we use that definition of priority, having the priority close to 1 means that almost every qualified employee will have to work at that job in order to cover the demand. When the priority is closer to 0, we have a lot more flexibility when assigning employees. If the priority is ever above 1, it means that the job's demand cannot be completely covered.

However, this formula for computing the priority has an important flaw. Consider two jobs j_1 and j_2, both requiring an employee's worth of time. If e is the only employee qualified for these jobs, the formula gives a priority of 1 for both j_1 and j_2, which would suggest that j_1 and j_2's demand could be covered. However, in reality, we can only cover one of these jobs's demand. It thus appears that this heuristic does not handle employees qualified for several high-priority jobs well, deflating these jobs' priority.

Improved version

Because of the existence of such imprecisions, another way to compute a job's priority is needed. To avoid the previous version's flaws, we will use a pre-assignment of the employees

to the jobs, in order to have a more realistic estimate of how many employees can actually work at a specific job. This pre-assignment will be a lot less constrained, in order to produce quick results. Specifically, we allow partial assignment, which allows to assign 1/3 of an employee to a job and 2/3 to another if it proves necessary. This kind of simplification is quite similar to using a linear relaxation in order to solve a MIP.

Formally, we introduce a number of weights $w_{e,j}, e \in E, j \in J$, which will represent which portion of the employee e is assigned to the job j. Since each employee is split, we have the constraint $\forall e \in E, \sum_{j \in J} w_{e,j} = 1$. Also, since employees may not work at jobs they are not qualified for, we also have $j \notin Q_e \implies w_{e,j} = 0$.

If we have a given value for the $w_{e,j}$, it is quite easy to compute how much manpower a given job is assigned. From that we are able to compute a more accurate job priority. Thus, we look for $w_{e,j}$ values allowing to cover as much as possible the job's manpower demands.

In order to do that, we used an algorithm inspired by gradient descent. First, we start with a uniform value for the $w_{e,j}$, to which we add a small noise. Then we execute the following loop for a limited number of iterations:

- First, we compute the coverage ratio of each job, if we assign the employees according to the current value of $w_{e,j}$. The coverage ratio is not defined for partial employees, but we naturally extend its definition to:

$$C_s(j) = \min(\min_{t \in H} \frac{\sum_{e \in E} w_{e,j} A_e(t)}{D_j(t)}, \frac{\sum_{e \in E} w_{e,j} W_e}{\int_H D_j})$$

  The values of $C_s(j)$ that we obtain indicate whether there is too much or too few employees assigned to j

- Then, for all $j \in J, e \in E$ we divide the value of $w_{e,j}$ by $C_s(j)$. Basically, jobs that have excess employees push those employees away, while jobs that require more employees pull those employees toward them.

- Lastly, we still have a normalization constraint to follow, so we normalize the $w_{e,j}$ by dividing them by $\sum_{j \in J} w_{e,j}$

That procedure usually converges rather quickly, but since it is fast we let it run for 20 iterations, just to be safe. Even with that precaution, it does not take much time. At the end of that loop, we take the inverse of the obtained coverage ratios in the first step as a job priority.

3.3.2 Actual assignment

After the priority of each job has been determined, the mapping assigning each employee to one specific job must be computed. As we previously mentioned, it will be created by assigning the employees one at a time. When there are no unassigned employees left, the mapping is complete.

First, let us notice that while the priority computed in the previous section gives us a good idea of which jobs need to be prioritized during the assignment, we should also consider whether a job is close to having enough employees to cover its demand when we decide to which job to assign an employee. To do so, we introduce a job's modified priority. In order to avoid confusion, we will use the term "intrinsic priority" when referring to the priority of a job as defined in the previous section.

$$P'_j = P_j(1 - C(j, f^{-1}(j))^s)^{s^{-1}}$$

$$s = 0.5 + 3C(j)$$

To obtain the modified priority P'_j, we multiply the intrinsic priority by a coefficient varying from 1 to 0 when the job's coverage ratio changes from 0 to 1. This is to reflect the fact that regardless of how prioritary a job is, the more its demand gets covered by the employees that are assigned to it, the less we have to care about how we will manage to fill it. The exact shape of decline of the multiplier is controlled by the sharpness parameter s. It is high when the job intrinsic priority is high, in order to have the multiplier close to 1 until the coverage ratio gets really high, and will start to fall down afterwards. It is low when the intrinsic priority is low as well, so that the multiplier decreases far earlier, which will allow employees qualified for that job to be taken away by jobs that have a high modified priority when the coverage ratio gets high enough.

Figure 3.2 shows the values of the multiplier for jobs with different priorities. The x-axis represents the coverage ratio of the job, and the y-axis represents the values of the multiplier P'_j/P_j. From top to bottom, the curves are a job with intrinsic priority 1, another job with intrinsic priority 0.33, and the last curve is for a theoretical job with an intrinsic priority of 0.

Now, we will assign employees to jobs by keeping track of two sets: the set of the jobs we still have to treat J', and the set of unassigned employees E'. They are initially equal to J and E respectively. Then, we use the following procedure until J' is empty:

1. Pick a job j from J' whose coverage ratio is minimal.

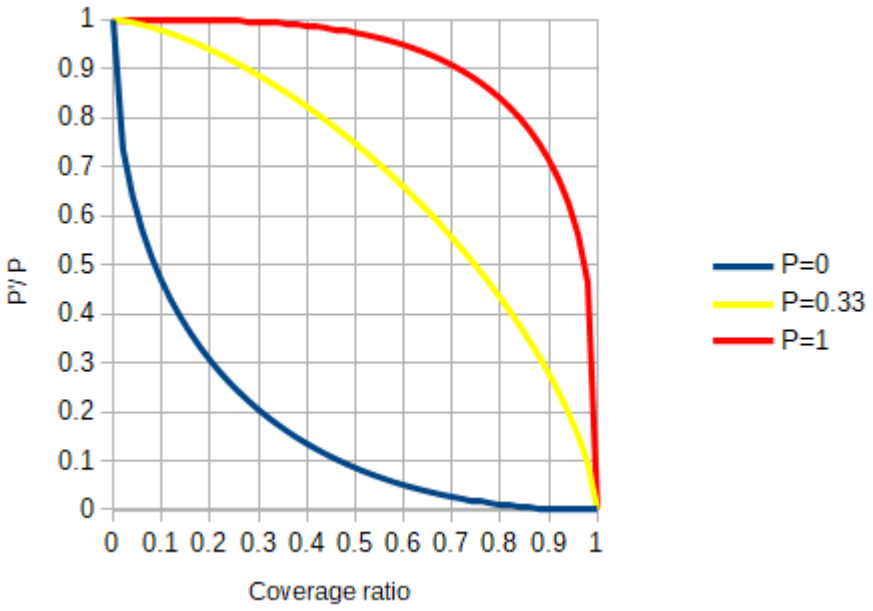


Figure 3.2 Plots of the multiplier curves for jobs with different intrinsic priorities

2. If $Q_j \cap E' = \emptyset$ or $C(j, f^{-1}(j)) \geq 1$, remove j from J' and go back to step 1.
3. From $Q_j \cap E'$, choose e such that it minimizes $r_e = \sum_{k \in Q_e \cap J'} P'_j$
4. Assign e to j, and thus remove e from E'.

When J' is empty, it is not guaranteed that E' is too, but the employees left in E' are only qualified for jobs that are already sufficiently covered, so it matters little how they are assigned. In our implementation of the algorithm, they were assigned greedily according to the possible jobs's coverage ratio.

That method of assigning employees to jobs has several advantages. First, by selecting the least covered job as the job to receive an employee, it tries to have a high minimum coverage ratio, which means that if it is impossible to cover the entirety of the demand for all jobs, the undercoverage will be spread amongst the jobs. It is generally a desired behaviour as it minimizes the number of temporary employees a company must hire should they decide to cover the demand at all costs. Second, the way we select employees favors employees that are not qualified for high priority jobs, as well as employees that are qualified for few uncovered jobs. That limits the impact of the two main issues we can get when computing an employee-job assignment: having undercovered jobs that have no qualified employees left unassigned, and having unassigned employees that are not qualified for any undercovered job.

3.4 Clustering the jobs

Now that the function $f : E \to J$ tells us which employees will be working each job, we proceed with the clustering, by first defining an affinity between the jobs, and then using that affinity to cluster them.

3.4.1 Choosing a relevant affinity

In order to define a relevant affinity between the jobs, we must consider the objective the clustering is meant to serve. In our case, it is two-fold. First, we want to minimize the effect of the mistakes we made while assigning the employees to the jobs in the previous section. Second, we want to produce clusters with a similar number of employees in order to minimize the time required to solve the subinstances. Since this part of the objective will be completed by choosing an appropriate clustering algorithm, we only need to consider the first part of the objective to construct our affinity.

The affinity between two jobs characterizes how much we want these two jobs to belong to the same cluster. To determine the affinity between jobs, we will first need to derive some intermediate values. Namely, we define:

$$W_{j_1,j_2} = \sum_{e \in Q_{j_1} \cap Q_{j_2} \cap (f^{-1}(j_1) \cup f^{-1}(j_2))} W_e \tag{3.3}$$

$$W_{j_+ \to j_-} = \sum_{e \in f^{-1}(j_+) \cap Q_{j_-}} W_e \tag{3.4}$$

$$W_j^+ = \min(\int_H \max(0, (\sum_{e \in f^{-1}(j)} A_e) - D_j), \sum_{e \in f^{-1}(j)} W_e - \int_H D_j) \tag{3.5}$$

$$W_j^- = \min(\int_H \max(0, (D_j - \sum_{e \in f^{-1}(j)} A_e)), \int_H D_j - \sum_{e \in f^{-1}(j)} W_e) \tag{3.6}$$

W_{j_1,j_2} represents how many hours of work could be moved around between j_1 and j_2. $W_{j_+ \to j_-}$ is the unidirectional equivalent of W_{j_1,j_2}, specifically representing how many work hours can be taken away from j_+ in favor of j_-. W_j^+ is how many work hours could be taken away from j while still meeting the demand. Conversely, W_j^- represents how many work hours are needed to adequately cover j's demand.

The affinity between two jobs j_1 and j_2 is then computed using a different formula for each of the following cases:

- If $C(j_1, f^{-1}(j_1)) < 1, C(j_2 f^{-1}(j_2)) \geq 1$ or $C(j_1, f^{-1}(j_1)) \geq 1, C(j_2 f^{-1}(j_2)) < 1$. Let's

call j_- the job that is insufficiently covered and j_+ the job that is sufficiently covered. Then, if there are employees associated with j_+ that are also qualified for j_-, we want to put these jobs in the same cluster so that the excess work hours that employees associated with j_+ can work are used to cover the shortage of employees working j_-. Therefore, the affinity between these two jobs is defined as $A(j_+, j_-) = \min(W_{j_+ \to j_-}, W^+_{j_+}, W^-_{j_-})$, which is an upper bound for the number of hours of undercoverage that could be fixed.

- If $C(j_1, f^{-1}(j_1)) < 1$ and $C(j_2, f^{-1}(j_2)) < 1$. In other words, if both jobs are undercovered, putting these two jobs in the same cluster is slightly useful. The first reason in favor of this is that the cost function of personnel scheduling problems sometime favors spreading undercoverage amongst different jobs instead of concentrating it. By putting j_1 and j_2 in the same cluster, we could allow their undercoverages to be balanced if there are employees associated with them that are qualified for both jobs. The second reason is that, depending on the final clustering obtained, it could be possible that one of the job's demand can actually be covered relatively easily. In this situation, some of the employees originally associated with that job could be transferred to the other one to fix its undercoverage. Therefore, we value putting these two jobs together at $A(j_1, j_2) = \lambda_3 W_{j_1, j_2}$, with λ_3 significantly lower than 1.

- If $C(j_1, f^{-1}(j_1)) \geq 1$ and $C(j_2, f^{-1}(j_2)) \geq 1$. In other words, if both jobs's demands are covered, putting them together is not really useful. However, if some of the employees are qualified for both jobs, it increases the number of feasible solutions to the personnel scheduling problem on the cluster, which might result in a better final solution. For example, fixing an undercoverage might require pulling so many employees from j_1 that it becomes undercovered itself. If j_2 has extra employees that happen to be qualified for j_1, its new undercoverage could be addressed. However, since it is a quite meager hope we shall not value it too much. Therefore, the affinity in this case is $A(j_1, j_2) = \lambda_4 W_{j_1, j_2}$, where λ_4 is quite lower than λ_3.

3.4.2 Clustering process

Requirements

In order to produce a high-quality partition of a personnel generation instance, the used clustering method must meet a few requirements:

- As mentioned in the introduction, since almost all known methods used to solve per-

sonnel generation problem have a superlinear time complexity, it is advantageous to make each subinstance of a similar size in order to lower the overall time needed to solve the master instance. Furthermore, having each subinstance be solved in a similar amount of time makes solving each subinstance in parallel more efficient.

Since the average time complexity of the method used to solve the subinstances is often hard to evaluate, the number of employees contained in all clusters will be approximately balanced in hope that it will be sufficient to have the subinstances solved in a similar time.

- The computation time is not our only concern. The quality of the results is also very important. The affinity we defined above is designed so that using it as a clustering criterion fixes the errors we made when we assigned the employees to the jobs, which should limit the loss of quality of the results. Therefore, we want to maximize the affinity between jobs that are in the same clusters.

Initial method

Our first attempt to devise an appropriate clustering algorithm produced inferior results. Nevertheless, it will be presented so that our mistakes will not be repeated.

Since we have a complex clustering problem involving a non-strict weighted balancing constraint, we were unable to use an off-the-shelf method, so we decided to try to use a MIP-based clustering method, in order to take advantage of its flexibility to add the balancing constraint.

We used a simple model, whose non-linear form is as follows:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j,l} a_{i,j} x_{i,l} x_{j,l} - \lambda_5 W_{max} \\
\\
\text{subject to} \quad & \sum_{l} x_{i,l} = 1 && i = 0, ..., n-1 \\
& W_{max} \geq \sum_{i} w_i x_{i,l} && l = 0, ..., N-1 \\
\\
& x_{i,l} \in \{0, 1\} && i = 0, ..., n-1; l = 0, ..., N-1 \\
& W_{max} \in \mathbb{R}^+
\end{aligned}$$

- $a_{i,j}$ is the affinity between the jobs i and j, as previously defined.
- n is the number of jobs we need to cluster.

- N is the desired number of clusters. As of now, it is chosen manually.
- λ_5 is a parameter controlling the relative importance of making good clusters and making balanced clusters.
- $x_{i,l}$ is a decision variable equal to 1 if the job number i belongs to the cluster number l and equal to 0 otherwise.
- w_i is the weight associated with each job. Since clusters with a similar number of employees are desired, it is equal to the number of employees assigned to the job i.
- W_{max} is a variable that is equal to the weight of the largest cluster.

The objective is to maximize the total affinity between jobs that belong to the same cluster. We apply a penalty proportional to the weight of the largest cluster in order to keep the cluster sizes approximately balanced. The constraints respectively ensure that a job belong to exactly one cluster and that W_{max} is indeed the size of the largest cluster.

As previously stated, this MIP is not linear, which would make solving it quite difficult. Fortunately, it can be linearized relatively easily. First, the textbook linearization of a product of binary variables was used. Using it, a variable $y_{i,j,l}$ is created and equal to the product $x_{i,l}x_{j,l}$. Then, as all values of $a_{i,j}$ are positive, the values of $y_{i,j,l}$ are the highest possible, so we can remove the constraints giving a minimal value for $y_{i,j,l}$. Lastly, by reformulating and combining the obtained constraints, the number of variables are reduced by replacing the $y_{i,j,l}$ by $y_{i,j} = \sum_l y_{i,j,l}$, giving the following equivalent formulation:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i,j} a_{i,j} y_{i,j} - \lambda_5 W_{max} \\
\\
\text{subject to} \quad & \sum_l x_{i,l} = 1 && i = 0, ..., n-1 \\
& y_{i,j} \leq 1 - x_{i,l} + x_{j,l} && i, j = 0, ..., n-1; i < j; l = 0, ..., (N-1) \\
& W_{max} \geq \sum_i w_i x_{i,l} && l = 0, ..., N-1 \\
\\
& x_{i,l} \in \{0, 1\} && i = 0, ..., n-1; l = 0, ..., N-1 \\
& y_{i,j} \in [0, 1] && i, j = 0, ..., n-1, i \neq j \\
& W_{max} \in \mathbb{R}^+
\end{aligned}
\tag{3.7}
$$

When we tested this method, two main shortcomings were found. The first one is rather simple. The branch-and-bound procedure commonly used to solve MIPs such as the one

we defined has a terrible complexity. Consequently, while very good computation times were obtained for small personnel scheduling instances, the performance was beyond terrible for the largest instance that was available: after 30 minutes of computation, the procedure was interrupted by an out-of-memory error. The second shortcoming is more subtle. The optimization objective is to maximize the sum of the affinities between jobs within the same cluster, with a penalty for the size of the largest cluster. Then, depending on the value of λ_5, we have one of these cases:

- If λ_5 is high, the balancing constraint will prevent any concessions on the sizes of the clusters in order to get clusters that have a better quality.
- If it is not, we end up with one giant cluster and possibly some tiny ones. The reason for that behaviour is that a large cluster's contribution to the objective is proportional to the number of edges it contains, which is itself close to the square of its size times the average value of the affinity. However, the size penalty we use is merely linear in cluster size. Therefore, the MIP solver will favor a very unbalanced solution containing a giant cluster, which does not suit our purposes.

3.4.3 Improved method

Since the previous clustering method is not satisfactory, a replacement based on hierarchical clustering was designed. The average linkage clustering criterion [20] was taken as a basis, and then modified to better fit our needs. The reason the average linkage was chosen is that the affinity we use has no property similar to the triangular equality: if a and b have high affinity, as well as b and c, nothing guarantees us that a and c also have high affinity. Since the complete linkage and single linkage criteria need some regularity in the metric used to be relevant, they would not be useful in our case.

First, we need balanced clusters, so we introduced a size penalty when computing the cluster merging criterion. To compute it, we first determine the target size of a cluster S_t, which is simply equal to the total number of employees divided by the desired number of clusters. Then, using that target size, we compute the normalized size S_n of the union of the clusters we are considering merging. The penalty we apply is then proportional to the fourth power of the normalized size. Formally, we compute the affinity penalty between two clusters B and C by the following formula, where N is the desired number of clusters:

$$S_t = \frac{|E|}{N}$$

$$S_n = \frac{|f^{-1}(B \cup C)|}{S_t}$$

$$P_s(B, C) = \lambda_7 S_n^4$$

The use of the fourth power in this penalty avoids the previous issue of favoring the appearance of a super-cluster, and also causes it to become quite severe once a cluster's size exceeds the desired size of the clusters, which will favor balanced clusters regardless of the exact values of the affinities between the jobs. Additionally, while the clusters are small, the size penalty is almost zero, which allows the clustering algorithm to favor maximizing the affinity so long as the cluster size is not a problem.

The choice of λ_7 depends on how much balanced clusters are valued over high-quality clusters, but it should be of the same order of magnitude as the average affinity.

Second, it is important to remember that the affinity between two jobs that we use was defined solely on the basis of what happens when these two jobs are the only jobs in a cluster. That approximation proves quite troublesome, as it can lead to overcompensation. Let us illustrate with a quick example. Assume that there is some job j that is somewhat undercovered. Then, the jobs that are overcovered and that have associated employees able to work the job j will have a high affinity with j. But, when we execute our clustering algorithm, the fact that we might have put enough of the aforementioned jobs in the same cluster as j to cover j's demand is never taken into account. Therefore, the clustering algorithm will try to put all the jobs that have a high affinity with j in j's cluster, while putting only a few would be sufficient to fix j's undercoverage.

In order to prevent this behavior, we will modify the average linkage criterion we used as a base. First, we will assign a weight $\omega_i \in [0, 1]$ to each job. It starts at 1 and decreases if the job's imperfections can be fixed using the current contents of the job's cluster. When computing the average affinity between two clusters B and C, we will then discount each individual affinity between two jobs by a factor equal to the product of the weights associated with these jobs. Formally, we use the following formulas to compute the value of our cluster merging criterion $r_{B,C}$:

$$r_{B,C} = \Big(\sum_{j_1 \in B, j_2 \in C} \omega_{j_1} \omega_{j_2} A(j_1, j_2)\Big) - P_s(B, C) \tag{3.8}$$

$$\omega_j = \frac{1}{1 + \lambda_6 c_j} \tag{3.9}$$

$$c_j = \frac{deg(j)}{\max(W_j^+, W_j^-)} \tag{3.10}$$

In these formulas, $deg(j)$ is the weighted degree of j in its current cluster. In other words, it is the sum of the affinities between j and jobs in the same clusters. If the job j is undercovered, c_j is the ratio between the extra work time that could be provided by other jobs in the cluster, and the actual number of work hours needed to fix the undercoverage. Conversely, if it is overcovered, it is the ratio between the work time that could be usefully given away and the extra work time available. In both cases, the higher it is, the closer we are to fixing j's imperfections. From c_j we derive the actual multiplier ω_j for the affinity values, depending on a parameter λ_6 controlling how fast a job's undercoverage or overcoverage is considered irrelevant. The value of λ_6 should be close to 1, in order to have a decently high multiplier while c_j is lower than 1. $r_{B,C}$ is then the value of the cluster merging criterion used in the hierarchical clustering algorithm.

3.5 Summary

To summarize, the main phases of this sequential algorithm are as follows:

1. Use a simplified assignment problem to compute the priority of each job. It represents how difficult each job's demand is to cover.
2. Use a greedy algorithm to actually assign each employee to a job. The objective of this phase is to have each job's demand coverable by the employees assigned to it.
3. Since errors were made in the previous phase, compute an affinity aimed to reduce the impact of these errors when a clustering based on it is computed.
4. Use a hierarchical agglomerative clustering algorithm to create clusters of jobs, using our custom cluster merging criterion.
5. The obtained clustering directly gives us a partition of the set of jobs, and the partition of the employee set can be trivially computed as the sets of employees that are associated with jobs belonging to the same cluster. These two elements allow us to partition the master instance in an efficient way.
6. The master instance is split into subinstances using the previously obtained partitions. Each subinstance is then solved using any applicable algorithm.
7. The final solution is obtained by computing the union of the solutions to the subinstances.

This is the extent of the sequential approach we created. We shall now present the results we obtained using it.

3.6 Results

We tested the sequential algorithm on five different instances of personnel scheduling problems. Some general data on these instances is available in the table 3.1. As a point of reference, we also used KRONOS's currently commercialized software to generate solutions to these instances. KRONOS's software uses an extra step to further improve the quality of an obtained solution. Once the subinstances have been solved and that the union of these solutions gives a candidate solution to the master instance, the undercoverages implied by that solution are catalogued, as well as the employees that could still work more. Then, the aforementioned employees are used to fix the undercoverages, to the greatest possible extent. In order to have a fair comparison, we will compare the obtained results with and without the use of this solution improvement algorithm. In the following, the unimproved solution will be called the raw solution, while the solution obtained after applying this method will be called the improved solution.

Table 3.1 – Testing instances used

Instance	Employees	Jobs	Planning Horizon (days)
large	350	320	7
med	106	42	7
mr	205	59	7
my	85	40	7
sm	127	44	7

In order to have decent results with the presented divide-and-conquer approach, the number of jobs should not be too low, in order to allow employees to potentially work at several different jobs. Therefore, as our smallest instances contain about 40 jobs, we chose to partition the instances into 10 subinstances in order to fit that constraint. The solutions to these subinstances were then computed using KRONOS's commercial software with its partitioning process disabled. Data regarding the obtained subinstances sizes is available in the table 3.2.

The actual results of using the partition generated by the sequential algorithm are available in the table 3.3. In this table, the first two lines refer to which solving algorithm the result is related, as well as whether KRONOS's solution improvement algorithm is used. Then, T is the total computation time necessary to produce the solution. When it is expressed in percents, it represents the increase or decrease of the computation time compared to the reference. T_p is the time required to generate the partition. T_s is the time needed to solve the obtained subinstances, as well as applying KRONOS's solution improvement algorithm, if

Table 3.2 – Size data regarding the partition obtained using the sequential algorithm.

Instance	Smallest Size	Average Size	Largest Size	Std Deviation
large	24	35	66	12.72
med	4	10.6	39	10.83
mr	10	20.5	41	9.66
my	0	8.5	35	10.45
sm	2	12.7	50	14.04

required. Lastly, the Cost column indicates the increase or decrease in solution cost compared to the corresponding reference.

Table 3.3 – Computation time and cost comparison for the solutions obtained via the sequential approach

Instance	Reference		Sequential							
	Raw	Improved	Raw				Improved			
	$T(s)$		$T_p(s)$	$T_s(s)$	$T(\%)$	Cost(%)	$T_p(s)$	$T_s(s)$	$T(\%)$	Cost(%)
large	114	274	26	430	300.0	-9.2	26	463	78.5	-9.5
med	106	139	2	101	-2.8	-6.1	2	222	61.2	-11.7
mr	314	336	14	109	-60.8	-69.6	14	138	-54.8	-64.4
my	116	152	1	192	66.4	-30.0	1	212	40.1	-47.6
sm	9	16	2	12	55.6	-5.3	2	22	50.0	-4.5

As we can see, our proposal for partitioning instances of the personnel scheduling problem gives us solutions that have at least a slightly better cost than our reference. In some cases, such as the 'mr' instance, our proposal significantly outperforms the reference, reducing the solution cost by around 66%. However, in terms of computation time, the performance of our sequential method is less impressive. While the time required to compute the partition is quite low compared to the reference's computation time, the time needed to solve the sub-instances derived from that partition is at best similar to the reference's time and at worst four times longer. The only exception to that is again the 'mr' instance, for which our results are just better across the board, both in term of cost and of computation time.

Whether the solution improvement algorithm is used does not seem to greatly change the global quality of our results. However, the results related to individual instances can change quite a lot depending on that decision.

Overall, when compared to the reference, the sequential algorithm trades a higher execution time for a lower cost. We see such a trade-off as favourable, but it seems that KRONOS's market studies indicate that the users of personnel scheduling software favor speed of execu-

tion quite highly.

CHAPTER 4 HOLISTIC METHOD

While the sequential method gives us quite decent results, there are still quite a few imperfections we would be glad to rid ourselves of. The most glaring one is that each employee is unequivocally associated with exactly one job. There is unfortunately no reason to believe that this is optimal, nor that it is even possible to do such an assignment without loss of quality. A simple example of that loss of quality is observed when the job's demand varies a lot during a week, which is a moderately common scenario. If there are one employee and 5 jobs that each require one employee during a different day of the week, fully assigning the employee to any of these jobs would leave the other jobs undercovered.

There is also the issue that every decision that is made is never reconsidered, so the slightest of mistakes made in the early steps of the algorithm could have far-reaching consequences. Removing that kind of liability would also be beneficial.

In order to hopefully address these issues, an alternative to the sequential method will be proposed, based on a completely different approach. First, a simpler problem that is closely related to the personnel scheduling problem will be defined. This simpler problem is then solved using a variant of the gradient descent algorithm. Lastly, the partition is deduced from the solution to this simpler problem. The results obtained using this approach will be compared both to the sequential approach and to KRONOS's reference.

4.1 Relaxation of the master instance

This method is inspired from the branch-and-bound procedure used to solve linear integer programs. In this procedure, the first step is to solve a simpler version of the integer program called the linear relaxation, which is the original program with the integrality constraints removed. The solution to that easier problem is then used as a basis to solve the harder original problem.

In our case, the integrality constraints on both assigning an employee to a job on a given day and assigning a job to a cluster will be removed. In other words, the real-valued variables $w_{e,j,p} \in [0,1]$ are introduced, representing a partial assignment of the employee e to the job j on the day p of the planning horizon. The clustering will also be included in our relaxed problem, by introducing the real-valued variables $x_{j,c} \in [0,1]$, representing the partial assignment of the job j to the cluster c. Then, the objective is to find some values of $w_{e,j,p}$ and $x_{j,c}$ such that:

- Each job is adequately covered by its associated employees.
- Each cluster is clearly defined. In practice, that means that each job is assigned to a single cluster, and each employee is only assigned to jobs assigned to the same cluster.
- The cluster's sizes are approximately balanced.

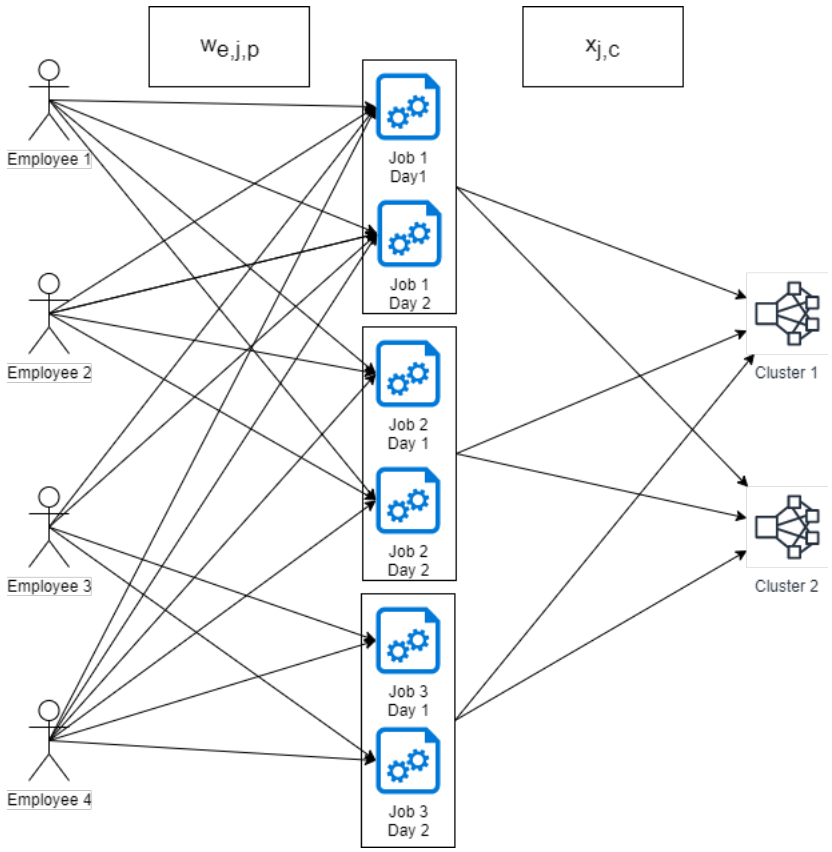


Figure 4.1 Diagram of the relaxed problem

If doing that is feasible, the obtained clusters should be a suitable partition of the master problem. To better visualize the problem, a diagram is available in the figure 4.1.

First, let us examine which constraints we are working under. On each day, each employee is split between an arbitrary number of jobs for which he is qualified. Therefore, $\forall p \in P, e \in E, \sum_{j \in J} w_{e,j,p} = 1$, where $P \subset \mathfrak{P}(H)$ is the set of all the days of the planning horizon, since regardless of how an employee is shared amongst jobs, he cannot work more than his normal workload. We must also have $\forall e \in E, p \in P, j \in J, w_{e,j,p} > 0 \implies j \in Q_e$, since no employee may work at a job for which he is unqualified. Similarly, each job might be shared between the different clusters, therefore $\forall j \in J, \sum_{c \in C} x_{j,c} = 1$, where C is the set of the desired clusters.

The values of $w_{e,j,p}$, and $x_{j,c}$ satisfying the aforementioned conditions are the feasible solutions to the relaxed problem. The set of all such solutions will be called F.

For any such feasible solution, it is possible to compute the coverage of each job by using the following formula, which is a natural expansion of the previously defined coverage ratio:

$$C_H(j) = \min_{p \in P}(C_p(j))$$

$$C_p(j) = \min(\frac{\sum\limits_{e \in E} w_{e,j,p} \frac{W_e}{|P|}}{\int\limits_{t \in p} D_j(t)}, \min_{t \in p} \frac{\sum\limits_{e \in E} w_{e,j,p} A_e(t)}{D_j(t)})$$

Also, it is quite trivial to compute the size of each cluster, which is equal to $W(c) = \sum\limits_{j \in J} x_{j,c} \sum\limits_{e \in E, p \in P} \frac{w_{e,j,p}}{|P|}$.

Using these results, the quality of a feasible solution can be estimated, both in terms of job coverage and of cluster balance. If you recall our objective, it is to find good solutions for which the clusters also clearly defined. Having clearly defined clusters means that the desired solutions must verify $\forall j \in J, c \in C, x_{j,c} \in \{0,1\}$ and $\forall e \in E, p \in P, \exists c \in C, \forall j \in J, w_{e,j,p} > 0 \implies x_{j,c} = 1$. An advantage of this model over the sequential formulation is that it does not require an employee to be entirely assigned to one particular job.

4.2 Solving the relaxation

Now, the objective is to find a way to compute a feasible solution that has good job coverage, whose clusters are well defined and balanced. In order to do that, a variant of the gradient descent algorithm will be used. The reason why the standard version of the gradient descent is not used is that it is quite difficult to compute the partial derivatives of a job's coverage ratio. Also, using standard gradient descent requires using a small step for modifying the values of $w_{e,j,p}$, and $x_{j,c}$, which would be slow. Therefore, the standard implementation is unusable. It could be possible to approximate the partial derivative by measuring how the coverage ratio changes when the value of a parameter is changed slightly. However, it would have to be done for every value of $w_{e,j,p}$ with $e \in E, p \in P$. Doing that for every job could be quite slow. On top of these issues, a solution that has well-defined clusters is required. Since standard gradient descent algorithms can produce any feasible solution, some way to get a final result that have well-defined clusters is required.

Therefore, specific knowledge of the problem will be used to adapt the gradient descent method to fit our needs. First, let us notice that, strictly speaking, finding an optimal

solution to the relaxed problem is not necessary. It is merely sufficient to find a solution that has the same clusters as the optimal ones. It seems reasonable to think that almost optimal solutions have a structure that is very similar to an optimal solution's structure, so there is a bit of leeway regarding the quality of the obtained solution. Second, since the gradient descent is not used in a black-box situation, it is possible to partially predict the effects of a modification of the values of $w_{e,j,p}$ and $x_{j,c}$.

With that in mind, the following algorithm was designed:

First, randomly initialize $w_{e,j,p}, e \in E, j \in Q_e, p \in P$ and $x_{j,c}, j \in J, c \in C$ uniformly between 0.8 and 1.2, then normalize these values. Then, for a fixed number of epochs, do the following steps:

1. For each job j, compute the value of the daily coverage ratio $C_p(j)$ under the current values of $w_{e,j,p}$ and $x_{j,c}$. Then, the new value of $w_{e,j,p}$ is computed using the following formula: $w'_{e,j,p} = \frac{w_{e,j,p}}{C_p(j)}$. That way, excess employees are pushed away from an over-covered job, and qualified employees are drawn towards undercovered jobs. Since the values of $w'_{e,j,p}$ are not necessarily normalized, they are then normalized.

2. Then, the values of $w'_{e,j,p}$ will be modified to get them closer to a situation where the clusters are clearly defined. To do so, how close an employee is to a given cluster will first be computed. It is done by defining $a_{e,c} = \sum_{j \in J, p \in P} \frac{w'_{e,j,p} x_{j,c}}{|P|}$. Then, for any employee e, the cluster he is closest to is determined, that is, the cluster $\bar{c}$ who maximizes $a_{e,\bar{c}}$. Then, the partial assignments of the employee is skewed to favor jobs pertaining to $\bar{c}$ and to penalize the other jobs. To do so, the following formula is used:

$$w''_{e,j,p} = (1 - \alpha) w'_{e,j,p} + \alpha \frac{x_{j,\bar{c}} w_{e,j,p}}{\sum_{j' \in J} x_{j,\bar{c}} w_{e,j',p}}$$

   This is simply a weighted average between the current value of $w_{e,j,p}$ and the normalized restriction of $w_{e,j,p}$ on the cluster $\bar{c}$. The weight α varies from 0 to 1 with the epoch. The exact values that are used are documented in figure 4.2.

3. Now, the clusters's sizes will be balanced. To do so, each cluster's current size will be computed. The size of a cluster c is easily computed using the formula $W(c) = \sum_{j \in J} x_{j,c} \sum_{e \in Q_j} \frac{w_{e,j,p}}{|P|}$. From this, the normalized size $\bar{W}(c)$ can be computed by dividing the cluster size by the desired size. If the normalized size is not between 0.8 and 1.2, the values of $x_{j,c}, j \in J$ are divided by $\bar{W}(c)$, in order to encourage the creation of approximately balanced clusters. Afterwards, the values of $x_{j,c}$ are normalized to conform to the constraint $\forall j \in J, \sum_{c \in C} x_{j,c} = 1$.

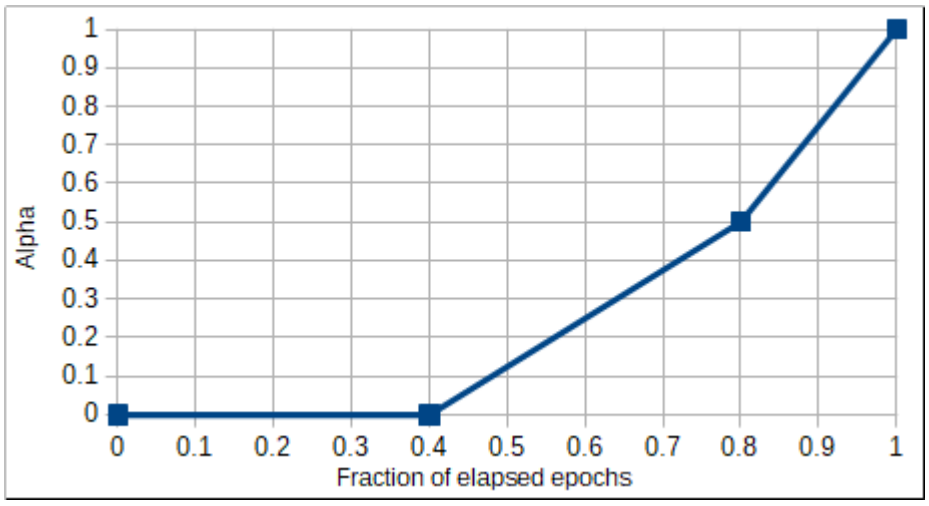


Figure 4.2 Values of α depending on the epoch

4. Then, the values of $x_{j,c}$ will be updated in order to improve the quality of the clusters. To do that, the value $a_{e,c}$ that we defined earlier will be reused as a measure of how close an employee is to a cluster. To estimate to which cluster a job should belong to, the values of $a_{e,c}$ will be evaluated for the employees associated with the job. Formally, $x'_{j,c} = \sum_{e \in E} a_{e,c} \sum_{p \in P} \frac{w_{e,j,p}}{|P|}$ is $x_{j,c}$'s new value. These values are then normalized to ensure that the $\sum_{c \in C} x_{j,c} = 1$ constraint is met.

5. Lastly, the values of $x_{j,c}$ are shifted in order to get closer to well-defined clusters. To do so, for each job $j \in J$, the cluster $\bar{c}$ that maximizes $x_{j,\bar{c}}$ is determined. Then, the new value of $x_{j,c}$ is:

$$x''_{j,c} = (1 - \beta)x'_{j,c} + \beta\delta_{c,\bar{c}}$$

This is a weighted average between the current value of $x_{j,c}$ and a discrete job-cluster assignment. The parameter β determines how quickly the shift towards a discrete assignment happens. It varies from 0 to 1 with the epoch. The exact values that are used are presented in figure 4.3

The values of α and β are carefully chosen. During the first epochs, they are both zero, in order to approach a satisfactory solution, that does not have well defined clusters. Then, β slowly climbs to start creating tentative clusters. Once clusters are more sharply defined, α starts to increase in order to bring the employee assignements closer to an acceptable solution. Since this changes the sizes of the clusters, the tentative clusters may be modified to better fit the new employee assignment. After that, both α and β are progressively brought to 1, to ensure that the obtained solution has well-

defined clusters.

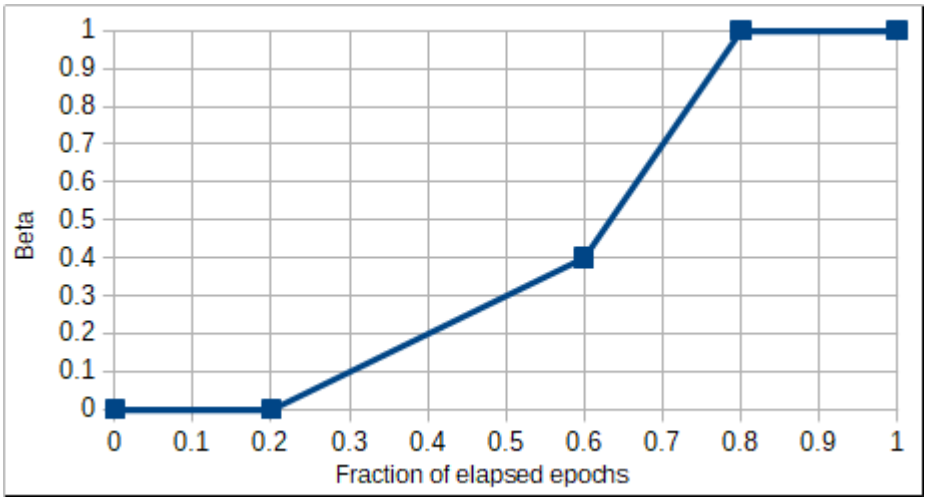


Figure 4.3 Values of β depending on the epoch

Once these steps are done for each epoch, the obtained clustering is extracted, which will be the partition used to divide the master scheduling problem. Formally, the jobs associated with cluster c are $\{j \in J | x_{j,c} = 1\}$ and the associated employees are $\{e \in E | \exists p \in P, w_{e,j,p} > 0\}$. Since $\alpha = \beta = 1$, these sets are indeed disjoint, and thus can be used to split the master instance into subinstances. As in the sequential approach, each subinstance is solved by an external algorithm, and the individual solutions are then aggregated into the final solution.

4.3 Results

In order to be able to compare the performance of the sequential approach and the holistic approach, the latter was tested in the exact same conditions as the former, on the exact same instances. The statistics related to the size of the partition's elements are presented in table 4.1, while the general results of the holistic method are available in table 4.2. To make it easy to compare the two proposed approaches, the table 4.3 compares the results obtained by both of the proposed approaches. The sequential approach is used as a reference, so positive values indicate that the holistic approach is slower or produces worse solutions than the sequential one. Conversely, negative values indicate better or faster results.

After looking at the table 4.1, one notices that subinstances with no employees are occasionally returned by the holistic algorithm. While it seems to be a liability, the computational results do not seem to be severely affected by it.

Table 4.1 – Size data regarding the partition obtained using the holistic algorithm.

Instance	Smallest Size	Average Size	Largest Size	Std Deviation
large	10	35	64	17.22
med	0	10.6	35	10.83
mr	0	20.5	77	23.49
my	0	8.5	24	9.13
sm	0	12.7	49	14.87

Table 4.2 – Computation time and cost comparison for the solutions obtained via the holistic approach

Instance	Reference		Holistic							
	Raw	Improved	Raw				Improved			
	$T(s)$		$T_p(s)$	$T_s(s)$	$T(\%)$	Cost(%)	$T_p(s)$	$T_s(s)$	$T(\%)$	Cost(%)
large	114	274	41	325	221.1	-12.7	41	396	59.5	-15.5
med	106	139	2	130	24.5	-15.3	2	139	1.4	-13.8
mr	314	336	61	751	158.6	64.9	61	749	141.1	93.0
my	116	152	1	102	-11.2	-47.6	1	98	-34.9	-28.8
sm	9	16	2	26	211.1	-4.5	2	27	81.3	-0.9

With one notable exception, the holistic approach produces better solutions than our reference, at the cost of an increased computation time. Unfortunately, it seems to give terrible results on the 'mr' instance, which is also the very instance on which the sequential method has stellar results. Given the extreme disparity in results, it is possible that the 'mr' instance has some unidentified special characteristic that is well handled by the sequential method, but poorly managed by the holistic one. On the other instances, the holistic method seems to produce results that slightly better than the ones obtained by the reference. However, these results are produced slightly more slowly than the reference when the solution improvement

Table 4.3 – Computation time and cost comparison between the solutions obtained via the sequential approach and the holistic approach

Instance	Raw		Improved	
	$T(\%)$	Cost(%)	$T(\%)$	Cost(%)
large	-19.7	-3.9	-10.6	-6.6
med	28.2	-9.8	-37.1	-2.4
mr	560.2	442.4	432.9	442.1
my	-46.6	-25.1	-53.5	35.9
sm	100.0	0.8	20.8	3.8

algorithm is used, and significantly more slowly if it is not. Thus, it seems that the holistic method benefits from being paired with KRONOS's solution improvement algorithm.

CHAPTER 5 CONCLUSION

In this chapter, the works done during this research master will be summarized. First, our main proposals and their results will be highlighted. Second, we shall point out their limitations, and propose a few topics of interest that may be investigated in future research.

5.1 Summary of Works

In this thesis, we considered how to best partition an instance of a personnel scheduling. Creating the sub-instances of the master instance is not difficult, but deciding how to exactly split it is non-trivial. Indeed, the jobs and employees should be distributed amongst the sub-instances in such a way that the sub-instances are of a similar size and that the solution of the split problem does not lose too much quality.

To achieve these objectives, two main methods were proposed:

- The sequential method: This algorithm requires computing a heuristic approximating a job's priority. Then, using this heuristic, each employee is heuristically and greedily assigned to a job in order to cover as much of each job's demand as possible. Afterwards, using that assignment, an affinity between each pair of jobs is computed. That affinity is then used by a clustering algorithm to produce an appropriate clustering of the jobs. Lastly, the partition is deduced from the obtained clustering.
- The holistic method: The previous proposal has many parts that have been arbitrarily chosen by ourselves. While its results are satisfactory, we sought a way to get rid of all these arbitrary choices and the imprecisions they bring. So, we created an easier but closely related problem which could be approximately solved using a customized gradient descent. Then, the partition can be extracted from the solution to that problem.

The results of our testing seem to indicate that both of these algorithms usually produce better solutions than the reference, while taking longer to do so. However, the holistic method seems to have reliability issues.

5.2 Limitations

These two methods have their share of limitations. The most glaring one is the holistic method's cluster size regularization's inability to do a decent job. While it avoids making

clusters of a ridiculous size, it produces clusters that are too unbalanced, which may result in increased computation times. It can even produce empty clusters sometimes. This is most likely due to the fact that in some situations, the effect of the size regularization will move the solution in an opposite direction to the effect of the other steps in the algorithms, which will render it effectively useless.

As the holistic method is based on a gradient descent, it is in theory quite sensitive to the solution initialization. While the problem does not look like is has many problematic local minima, we have no proof that it is indeed true.

Another general issue with our work is that many constants are chosen (such as the various λ, the shapes of α and β...) by hand after a few tests. Nothing guarantees that the chosen values are optimal, so it would be interesting to create a larger training set and use it to compute optimal values for these constants.

5.3 Future Research

A very trivial avenue of future research would be fixing the aforementioned limitations of our proposals.

Apart form that, we noticed that when the job priority is computed at the beginning of the sequential algorithm, it was done by computing an optimal set of $w_{e,j}$ that induces a good coverage of the demand. This set can be used to define a bipartite graph, whose vertex set is $E \cup J$, and the edges are valued with the values of $w_{e,j}, e \in E, j \in J$. Using that graph, one could try to use some clustering algorithm to directly obtain the partition. The main advantage of this formulation is that we have more off the shelf clustering methods we can use. Since balancing the sum of the number of jobs and the number of employees among the clusters is reasonable, we can now use balanced clustering methods such as those described in [18]. However, we would at least have to adapt the cost function of such a clustering to ensure that jobs are adequately covered. Designing an algorithm based on such a formulation could be an interesting continuation of our work.

Additionally, the presented algorithms were designed with the assumption that only the most basic data is available to compute the partition, in order to have algorithms usable on many different scheduling problems. If we relax that hypothesis, additional data could help creating a better partition. Such data could include the preferences of the employees, an history of the solutions found for similar instances, the relative importance of covering each job's demand or a non-deterministic demand.

REFERENCES

[1] L. C. Edie, "Traffic delays at toll booths," *Journal of the Operations Research Society of America*, vol. 2, no. 2, pp. 107–138, 1954. [Online]. Available: https://doi.org/10.1287/opre.2.2.107

[2] G. B. Dantzig, "Letter to the Editor—A Comment on Edie's "Traffic Delays at Toll Booths"," *Operations Research*, vol. 2, no. 3, pp. 339–341, August 1954. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v2y1954i3p339-341.html

[3] H. Karloff, *The Simplex Algorithm*. Boston, MA: Birkhäuser Boston, 1991, pp. 23–47. [Online]. Available: https://doi.org/10.1007/978-0-8176-4844-2_2

[4] 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.

[5] T. Aykin, "Optimal shift scheduling with multiple break windows," *Management Science*, vol. 42, no. 4, pp. 591–602, 1996. [Online]. Available: http://www.jstor.org/stable/2634390

[6] I. Addou and F. Soumis, "Bechtold-jacobs generalized model for shift scheduling with extraordinary overlap," *Annals of Operations Research*, vol. 155, no. 1, pp. 177–205, 2007.

[7] M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau, "Formal languages for integer programming modeling of shift scheduling problems," *Constraints*, vol. 16, pp. 54–76, 01 2011.

[8] M. V. Lawson, *Finite automata*. CRC Press, 2003.

[9] D. Attia, R. Bürgy, G. Desaulniers, and F. Soumis, "A decomposition-based heuristic for large employee scheduling problems with inter-department transfers," *EURO Journal on Computational Optimization*, vol. 7, no. 4, pp. 325–357, 2019.

[10] S. Mohan, "Scheduling part-time personnel with availability restrictions and preferences to maximize employee satisfaction," *Mathematical and Computer Modelling*, vol. 48, no. 11-12, pp. 1806–1813, 2008.

[11] R. Stolletz and J. O. Brunner, "Fair optimization of fortnightly physician schedules with flexible shifts," *European Journal of Operational Research*, vol. 219, no. 3, pp. 622–629, 2012.

[12] C. Froger, "Mise à jour des horaires de personnel travaillant sur des quarts," Ph.D. dissertation, École Polytechnique de Montréal, 2015.

[13] H. Michon-Lacaze, "Élaboration de quarts de travail robustes aux perturbations de courte durée," Ph.D. dissertation, École Polytechnique de Montréal, 2016.

[14] A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*, vol. 153, no. 1, pp. 3 – 27, 2004, timetabling and Rostering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S037722170300095X

[15] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck, "Personnel scheduling: A literature review," *European Journal of Operational Research*, vol. 226, no. 3, pp. 367 – 385, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221712008776

[16] C. Hennig, M. Meila, F. Murtagh, and R. Rocci, *Handbook of cluster analysis*. CRC Press, 2015.

[17] L. H. N. Lorena, M. G. Quiles, L. A. N. Lorena, A. C. de Carvalho, and J. G. Cespedes, "Qualitative data clustering: a new integer linear programming model," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[18] L. R. Costa, D. Aloise, and N. Mladenović, "Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering," *Information Sciences*, vol. 415, pp. 247–253, 2017.

[19] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[20] R. Sokal, C. Michener, and U. of Kansas, *A Statistical Method for Evaluating Systematic Relationships*, ser. University of Kansas science bulletin. University of Kansas, 1958. [Online]. Available: https://books.google.ca/books?id=o1BlHAAACAAJ