TÉCNICO LISBOA

Application of Scheduling Techniques to Teacher-Class Assignments

Pedro Miguel Andrade Ferreira

Dissertation submitted to obtain the Master Degree in

Information Systems and Computer Engineering

Supervisors: Prof. João Emílio Segurado Pavão Martins
Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. João Emílio Segurado Pavão Martins
Member of the Committee: Prof. Maria Inês Camarate de Campos Lynce de Faria
Dr. Ricardo Lopes de Saldanha

October 2015

Acknowledgments

A todos os que me ajudaram e apoiaram nesta etapa do meu percurso académico, gostaria de endereçar o meu sincero agradecimento. Em especial aos meus orientares, Prof. João e Prof. Luísa, pela paciência, apoio e orientação ao longo destes meses. Muito Obrigado!

Abstract

Nowadays, it is more important than ever to be as efficient as possible. At Instituto Superior Técnico, one task that is very time consuming is assigning professors to the courses as preparation to the following academic year. Teaching Service Distribution, as it is called, is one of the steps that must be accomplished to get the next academic year ready. We created a system that helps carrying out this task. Our system is divided into 3 components: Web-platform, database and TeachOpt Solver.

TeachOpt Solver is responsible for the automatically assignment the professors to courses. Literature shows that is it not common to have this task performed by itself. However, it reveals that there are two main areas and techniques that are used to perform this task: Operational Research and Artificial Intelligence techniques. We tested both techniques.

Operational research techniques consist on solving Integer programming model instances. Artificial Intelligence techniques are based on Local Searches. From the tests we performed, we concluded that Operational Research techniques performed better. They perform quickly to generate high quality solutions. Even though the solutions generated may need some manual adjustment, the system dramatically reduced the time it takes to have all professors assigned to courses.

Keywords: Teaching Service Distribution, Operational Research, Artificial Intelligence, Resource Assignment, Local Search

Resumo

Nos dias de hoje, é fundamental ser-se o mais eficiente possível. No Instituto Superior Técnico, uma das tarefas que mais tempo consome na preparação do próximo ano letivo é a afetação de professores às unidades curriculares. A distribuição do serviço docente, denominação dada a esta atribuição, é um dos processos que tem que ser concluído para que tudo esteja pronto para o começo do ano letivo. Assim criamos um sistema com o objetivo de ajudar à realização desta tarefa. Este sistema está divido em 3 componentes: Plataforma Web, Base de dados, Solver TeachOpt.

O Solver TeachOpt é a componente responsável por atribuir automaticamente os professores às unidades curriculares. A Literatura Cientifica mostra que não é comum ter esta tarefa realizada por si só. No entanto, revela que existem duas grandes áreas e técnicas que são utilizadas para realizar esta tarefa. Nós testamos tanto técnicas de Investigação Operacional, bem como técnicas de Inteligência Artificial.

As Técnicas de Investigação Operacional consistem em resolver instâncias de modelos de programação inteira. Já as técnicas de Inteligência Artificial são baseadas em procuras locais. A partir dos testes que realizamos, concluiu-se que as técnicas de Investigação Operacional obtiveram o melhor desempenho. Estas conseguem rapidamente gerar soluções de alta qualidade. Apesar das soluções geradas poderem precisar de alguns ajustes manuais, o sistema reduz drasticamente o tempo que leva a fazer a toda a distribuição do serviço docente.

Palavras-chave: Distribuição do Serviço Docente, Investigação Operational, Inteligência Artificial, Afetação de Recursos, Procura Local

Contents

List of Figures

List of Tables

List of Acronyms

DEI Department of Computer Science and Engineering

IST Instituto Superior Técnico

TSD Teaching Service Distribution

CS Course Scheduling

ILP Integer Linear Programming

OR Operational Research

AI Artificial Intelligence

CSP Constraint Satisfaction Problem

CP Constraint Programming

Chapter 1

Introduction

Schools, universities, and other educational institutions, in preparation of each academic period, have a complex and sometimes-difficult task in hand: the allocation of the faculty team and rooms to the courses to be taught. Planning each academic period involves several tasks such as forecasting students' enrollments, predicting students' choices for optional courses, scheduling courses, assigning the faculty team to each of the courses, and scheduling exams. Each school has its own way of planning the academic period. Some schools prefer to do some of these steps together. For instance, it is common to have Course Scheduling (CS), which is the assignment of courses to a time-slot and room, and Teaching Service Distribution (TSD), which is the assignment of professors to courses, solved in just one-step. However, all of them can be solved separately.

At Instituto Superior Técnico (IST), CS and TSD are solved as separated processes. These two processes are separated even if they run concurrently and share some data. If we considered the fusion of those two processes, it will lead to a complex problem. It is not easy to schedule isolated courses from one department as, it is not possible to schedule the courses from the other departments, since a degree is composed by courses of different departments. One change on the schedule of a course of Department A may lead to a complete reschedule of the courses of Department B. Of course, there are ways to mitigate the problem, but we just want to build a tool to help the TSD process. It is not our intention to redefine how IST plans each academic period.

This thesis focus on the TSD of Department of Computer Science and Engineering (DEI), IST. To better understand the process of assigning the faculty team to the available courses, in the following sections, we describe how it is performed.

1.1 Teaching Service Distribution Problem

As we concentrate our efforts on the TSD problem, there are some fundamental concepts that we need to have in mind. This problem consists on assigning a set of professors to a set of classes. Although it looks like a straightforward process, we must take into consideration that each professor has qualifications, that indicate which classes he/she may teach, and also has a number of credits or

"working hours", that indicate how many classes he/she will teach.

We must also consider that in each course, there are, typically, two types of classes, recitations and lectures and each type has a limit on the number of students that may attend to one session. As, every student must attend both types of classes, there will be multiple sessions of the same type of class.

1.2 Actual Method

Today, at DEI in IST, the process of generating the TSD is made by hand, using an excel file. This file aggregates all the information about the available professors and courses.

As an example, Figure 1.1 is a screenshot of an excel file used to generate the TSD at DEI. Each column represents a course, and each line a professor. The values in blue represent how many hours that professor must teach in that course. There are also formulas that automatically calculate the difference between desired teaching credits and the effective teaching credits and other formulas that indicate if all classes are covered.

The person in charge for this task, by hand, tries to satisfy all the professors' preferences and constraints.

It is possible to identify several steps in this process:

1. All the information about the forthcoming courses is reviewed, as well as the estimation of enrolled students in each course.

   The person in charge of the TSD checks which courses will be available in that particular year, considering that degrees can be restructured or other situations that can force a specific course not to be available or the addition of new courses. It is also here that, for each course and having in mind the previous years, forecasts of the number of enrolled students are made in order to calculate how many course sections that course must have and how many credits/hours that specific course will value to its teachers, despite the number of hours that he/she will have.

2. Professors are asked about how many hours they have to teach and how many credits they must have in that year.

   Each professor indicates how many credits he/she must have by the end of the year. The teaching load may change due to sabbatical leave or because the professor taught extra hours in the previous year. He/She can also specify each courses/classes he/she would like to teach. A professor, according to their rank, must reach, each year, a certain number of credits. Management positions, Master and PhD students contribute to decrease that number of credits, but most of those credits are obtained by teaching classes. There is a one-to-one relation between credits and teaching hours, but under certain conditions, for example, upper level courses with low student enrollment, one hour can be less than one credit.

3. The person in charge of the TSD gathers all the information and, by hand, tries to assign all classes to professors in order to fulfill all the requirements.

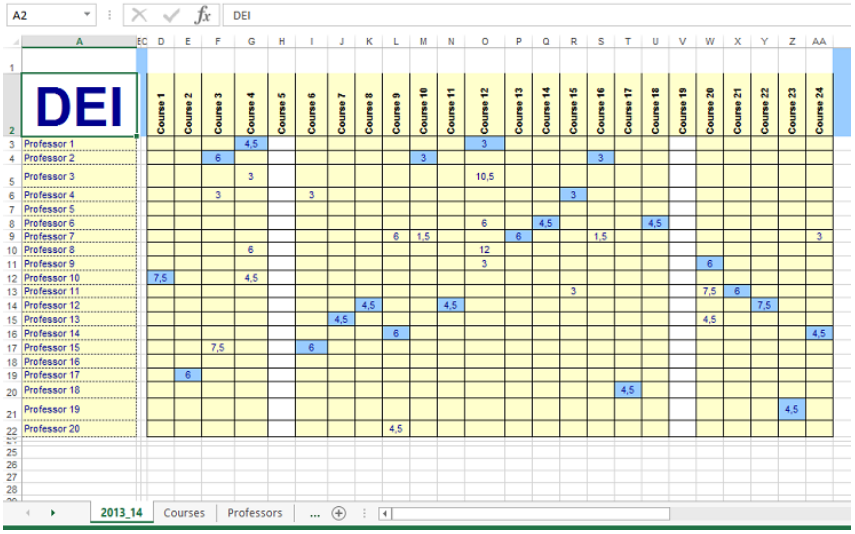

Figure 1.1: Example of the Excel file used.

This is the most complex step. There is no method for this task. It is performed on a trial-and-error basis. Mainly, we can identify that a first iteration is made by making all possible assignments considering the professors' preferences. When a conflict arises there is no method to solve it, just trial-and-error. Adjustments are made and when a complete solution is achieved, the process stops. Although most of the time a solution is found, there is a possibility that the solution does not exist, that is, at the end of the process, not all classes are assigned or one professor does not have the desired teaching credits. In this case, several scenarios have to be considered such as hiring part-time professors or relaxing the number of credits each professor must have.

4. In the last step, after students have registered for courses, the number of enrolled students are checked and the previous forecast numbers are now changed to the real ones. Finally, the calculation of the professors' credits in that year can be made.

The first three steps can be made without taking into account the actual number of students enrolled but the final step can only be made after that.

The overall process can be very time consuming. It takes several work-hours and several iterations to have a TSD that complies with all the requirements and satisfies as much as possible the preferences expressed by each professor.

1.3 Goals

This thesis aims at automating the TSD process by building an on-line platform where professors could introduce how many credits/hours they must have and specify which classes they prefer to teach, and in the end all the assignments are made according to the stated preferences and restrictions. We had analyzed the suitable existing scientific work and adapted it to our problem.

1.4 Thesis Contribution

The work done in this thesis led to the following contributions:

- Analysis of the literature about previous work done about TSD;
- Development of a software that can be used to automatically generate TSD;
- Development of a web interface to insert some essential data and visualize the generated TSD.

In a first evaluation, results show that it is possible to have a system to automate the assignment of professors to courses and even have better results than a manual assignment.

1.5 Outline

This document is organized as follows:

- **Chapter 2** describes the related work about the subject of this thesis;
- **Chapter 3** details important aspects of the problem, as well as the theoretical description of the techniques used;
- **Chapter 4** describes the system components and the architecture of the system;
- **Chapter 5** describes the implementation choices and the chosen technologies;
- **Chapter 6** details the evaluation tests that were made as well as the corresponding results;
- **Chapter 7** summarizes the work developed and future work.

Chapter 2

Related Work

The problem that we address in this thesis is classified as a resource scheduling problem. A *resource scheduling problem* consists in finding out combinations of tasks or activities, subjected to a certain number of constraints, and assigning them to resources according to certain rules and constraints. Both the combinations of tasks and the resources assignments aims at optimizing some goals. Resource scheduling is an active area of research, mainly in the areas of transportation and job shop scheduling. Resource scheduling may be looked under the perspective of a *decision problem*, where the goal is to find out whether it is possible to assign the available resources to the required tasks or under the perspective of an *optimization problem*, where the goal is to find out the best possible assignment of the available resources to the required tasks. In this thesis, we are concerned with the optimization problem.

In this chapter, we overview the resource scheduling problem under two perspectives: techniques and applications.

2.1 Techniques

Operational Research (OR) and Artificial Intelligence (AI) are the fields of research that most actively contribute to the solutions of resource scheduling problems. In this section, we present some models and algorithm that are used to solve scheduling problems.

2.1.1 Operational Research

Linear programming (Dantzig, 1964) is an OR method used to optimize an objective function in a mathematical model whose requirements are represented by linear equations. Integer Linear Programming (ILP) (Williams, 1980) is a variation of linear programming in which some or all the variable are restricted to be integers.

ILP is the most used contribution from OR in the solution of resource scheduling problems. It is possible to describe a wide range of problems using only mathematical expressions. ILP have problem-independent techniques that can solve any specified problem. For most problems, those techniques produce optimal solutions. However, under certain conditions, the computing time or space necessary

to produce such solutions is not available and the problems are relaxed and those techniques adapted in order to decrease the time needed to produce a solution. In this case, an optimal-enough solution is produced instead of an optimal one.

Most of solutions proposed in the literature about OR and resource scheduling problem is based on formulating an ILP model, which is a way of modeling a problem using mathematical expressions, that correctly modulates the real problem and improved techniques to solve those models with a huge number of variables. The techniques that are applied are independent from the model itself. A common approach is to specify the model is based on a set partitioning problem where we have a set V that represents our universe and a set S composed by subsets of V. The goal is to find a subset of S that has all elements of V a certain number of times.

Consider that we have a decision variable represented as follows:

$$x_{ar} = \begin{cases} 1 & \text{if allocation } a \text{ is assigned to resource } r \\ 0 & \text{otherwise} \end{cases}$$

and two parameters:

$$A_{art} = \begin{cases} 1 & \text{if allocation } a \text{ for resource } r \text{ has the task } t \\ 0 & \text{otherwise} \end{cases}$$

$$K_{ar} = \text{the value of assigning the allocation } a \text{ to resource } r$$

The basic model for a set covering problem is:

$$\text{minimize} \sum_{r=0}^{n} \sum_{a \in \Omega_r} K_{ar} x_{ar} \tag{2.1}$$

$$\text{subject to} \sum_{r=0}^{n} \sum_{a \in \Omega_r} A_{art} x_{ar} = b_t \qquad t = 1, 2, 3, ...m \tag{2.2}$$

$$\sum_{a \in \Omega_r} x_{ar} = 1 \qquad r = 0, 1, ..., n \tag{2.3}$$

$$x_{ar} \in \{0, 1\} \qquad r = 0, 1, ..., n \quad \forall_a \in \Omega_r \tag{2.4}$$

where:

$m =$ number of tasks

$n =$ number of resources

$\Omega_p =$ set of all feasible allocations for resource p.

Expression 2.1 represents the objective function that we want to maximize/minimize (A factor K_{ar} is multiplied by each possible assignment X_{ar}). Expression 2.2 (a constraint) ensures that each task is

assigned by exactly b_t resources. Expression 2.3 (a constraint) guarantees that only one assignment is selected per resource. Expression 2.4 (a constraint) forces the variables x_{ar} to be 0 or 1 depending on a assignment being selected or not.

A simpler model corresponds to a generic resource allocation. The decision variable X_{rt} is responsible for assigning each task to a resource. The 2D-matrix X is the basis for each constraint. The generic model is as follows:

$$\text{minimize } \sum_{r=0}^{n}\sum_{t}^{m} K_{rt}X_{rt} \tag{2.5}$$

$$\text{subject to } \sum_{r=0}^{n} X_{rt} = b_t \qquad t = 1, 2, 3, ...m \tag{2.6}$$

$$X_{rt} \in \mathbb{N} \qquad r = 0, 1, ..., n \qquad t = 1, 2, 3, ..., m \tag{2.7}$$

where:

$m =$ number of tasks

$n =$ number of resources

Comparing this model with the previous one, expression 2.5 represents the same as expression 2.1. Constraint 2.2 is equivalent to 2.6. Constraint 2.7 and 2.4 are used for the same propose.

A faster way to obtain a solution for the models that we presented earlier in this section is by solving the problem without the integer constraints, i.e, allowing the variable to take fractional numbers. And then, by rounding the values obtained of the problem variables, we obtain an integer solution for the ILP. This fast method does not ensure optimality nor admissibility of the solution. Sometimes this solution may be enough for certain problems but it may not be acceptable for our problem to get a non-optimal solution.

Branch and bound is one of the algorithms that are used to obtain an optimal solutions for ILPs. This algorithm relaxes the integer constraints (2.7, 2.4) and then, based on the obtained solution (without the integer constraints), introduces new constraints in order to limit the range of variables. This repeats until it reaches the integer solution (Lawler and Wood, 1966). One factor that decreases the performance of such algorithm is the number of variables.

The first model presented in this section assumes a previous generation of all feasible allocations for each resource. This job can be very time-consuming and may produce millions of allocations. The huge number of generated allocations will result in a large model which will be more complex and time-consuming to resolve. To address this problem, a technique called column generation is used to avoid enumerating schedules at once. They are produced as needed.

These two methods (branch-and-bound and column generation), when combined that is named Branch-and-Price, allow to focus the search and to avoid bad paths. With this approach they can solve

reasonable sized problems in a few hours by not enumerating all schedules at once. Several strategies can be employed for the generation of columns.

2.1.2 Artificial Intelligence

AI offers different methods to solve resource scheduling problems. It tries to adapt generic ideas in order to build/search a solution. It is possible to distinguish two main generic concepts about how to obtain a solution: constructive approaches build a solution from scratch and reparative approaches that improve or repair previously generated solutions.

Searches that use a constructive approach, iteratively create a solution by enumerating all the possible changes that can be applied to a at the given state. It all starts with an empty state. At each step of the search, a new state is built from the previous one for each small change that can be made to the current state and stored in memory. Then a state is picked from those that are in memory and the process repeats. This process constructs a search tree. Each leaf node is a state and each arc between node represent the changes that were made to obtain the following state. The search finishes when a solution is found. In searches using constructive approaches, it is not only possible to discover a solution as well as all steps that lead to that solution. Constructive search algorithms do not have a global perception of the problem or the solution that is being built. It is only possible to look at what is on the state at that particular moment, and sometimes an estimate of what is left. If a wrong decision is made at an early stage of the search, it may be difficult to recover from that mistake. On the other hand, in the OR methods, it is only possible to obtain the final solution without following how the solution was constructed and they have always a global perception of the problem.

Due to the fact that resource scheduling problems have a huge number of possible solutions as well as a huge number of combinations, constructive searches are not suitable to solve them. The amount of space and the time required by constructive searches is dependent on the depth (number of states that are needed to build a complete solution) where the solution can be found on the search tree. It is also dependent on the average branching factor (average number of states that are created at each step of the process) that tend to be huge in resource scheduling problems. Therefore, searches that use a constructive approach usually fail in resource scheduling problems.

Local searches and genetic algorithms are the main methods that are focused in the AI literature related to resource scheduling problems because they can produce good results.

Local Searches

Local searches are widely used in this resource scheduling of problems. Local searches start from a partially built solution and little changes are made at each step, also known as moves, knowing that a better solution may be produced after a certain number of steps. Local searches do not have any optimal guarantee, but it can quick produce good enough results. With these techniques, we may have more control over the overall process (Van den Bergh et al., 2013).

In local searches, at every step at least one complete solution is kept in memory, so, at any time, if we

stop the search, a possible solution can be presented. Local searches are used as a method to improve solutions' quality. At each step, changes are made in order to improve or at least maintain quality in the following steps. After certain number of steps or when a specific criterion is met, the search is stopped and the best solution found so far is returned.

Local searches are commonly tested to solve scheduling problems (Van den Bergh et al., 2013) and used to compare with other techniques. It can even produce better results than OR (Maenhout and Vanhoucke, 2010). The following local searches are the most common and widely used:

- Hill Climbing: It is a basic local search. At each step only moves that produce immediate improvements are allowed. This straightforward approach can be easily caught in local optima. In some cases, the solution obtained is very far from the optimal solution.
- Simulated Annealing: In order to avoid local optima that may happen in hill climbing, this search uses a strategy that allows worsen the quality of the solution. After a certain number of steps the probability of accepting a bad move decreases in order to intensify the search (Kirkpatrick et al., 1983).
- Tabu Search: This search keeps in memory a certain number of past moves/solutions (tabu list) in order to avoid repetitions. If a move or solution is in the tabu list, it cannot be considered has a possible solution. Each member of the tabu list as an expiration period. After this period that move/solution can be considered again (Glover, 1989).
- Variable Neighborhood Search: A procedure that tries to explore different neighbors of a initial solution in order to cover a wide search space

(Lü and Hao, 2012) proposed a local search that can change its behavior during the search process, giving rise to adaptive neighborhood search is a search that, according to its perception of the success, can change the strategy that is applied. At the beginning of the solution process, it uses a constructive heuristic to obtain an initial solution that will be improved (Lü and Hao, 2012). Afterwards, it uses three different strategies to improve the quality of the solution:

1. *Intensive Search*: From the actual best solution, a Tabu Search is performed in order to improve that solution to the maximum.
2. *Intermediate Search*: At each iteration, it selects a subset of resources and then it picks the best option considering just the subset.
3. *Diversification Search*: It is similar to Intermediate Search, except that it chooses randomly one of the options that improves the solution for not violating at least one of the constraints.

Each strategy is chosen based on the improvements of the objective function at each step. It first starts with the Intensive Search in order to quickly improve the objective function. When a local optima is found, it changes to the Intermediate Search and even if it happened again with this strategy, Diversification Search is applied.

As shown in (Lü and Hao, 2012), the correct selection of multiple possible changes (moves) of a solution can improve results obtained as it allows to cover a much larger search space.

Genetic and Evolutionary Algorithms

Another techniques that are being used to solve resource scheduling problems are Genetic algorithms. These techniques try to incorporate ideas from biology and how combinations are made in the natural world to optimize a solution. The main idea is that good characteristics of a solution are kept, and bad characteristics are dropped.

A hybrid scatter search was proposed in (Maenhout and Vanhoucke, 2010). This is an evolutionary approach that showed to produce better results than variable neighborhood search and the a Branch-and-Price procedure (branch-and-bound with column generation (Barnhart et al., 1998)). It could produce solutions 3% better than variable neighborhood search and up to 30% better than the Branch-and-Price.

This algorithm is based on the basic concepts of the canonical genetic algorithm (Whitley, 1994). *Selection* and *Recombination* are the basic ideas that any genetic algorithm must have. At each step a subset of the actual population is chosen based on same fitness criteria and them those elements in the subset are combined together in order to produce the next generation of the population for the next iteration.

The generic pseudo-code of such search is:[1]

Algorithm 1 Scatter Search

```
1: procedure SEARCH
2:     Diversification Generation Method
3:     while Stop Criterion not met do
4:         Subset Generation Method
5:         Solution Combination Method
6:         Improvement Method
7:         Reference Set Update Method
```

Several methods for steps 2, 4, 5, 6, 7 were tested in (Maenhout and Vanhoucke, 2010), but the most informed ones, reveled to perform better.

Step 2 is an initialization method: it is where the initial population is generated. This population is then divided into two different sets, Set_1 and Set_2. Set_1 has k_1 best elements, and Set_2 has the k_2 most diverse elements. They tested three different approaches, one completely random, a second one with an heuristic and the last is a mix of the two first ones, one half is generated randomly and the rest with a help of an heuristic. This last approaches revealed to be better to the overall search, helping to produce solutions up to 3% better than the other two approaches.

Step 4 is responsible to pick which two of the solutions in the Sets will be combined later. Previous work cited in the paper revealed that combining elements of the same set stimulates intensification and combinations from different sets stimulates diversification. They showed that if we first select which sets we want to combine ($Set_1 \times Set_1$ or $Set_1 \times Set_2$ or $Set_2 \times Set_2$) according to pre-established probabilities

[1] From (Maenhout and Vanhoucke, 2010)

and then within each set the best element is picked according to the characteristics of the set, better results were obtained.

At Step 5, the two elements picked in the previous step will be combined. The best approach was a combination of the best ones and each method is selected based on a probability. Those best methods were the ones that have some problem-specific knowledge and the traditional ones

Step 6 revealed to be the most important one. The absence of this step can lead to a decrease of quality of about 60% on the best solution found.

The final Step (7) allows the sets 1 and 2 to be updated. The most effective strategy is the update of those sets immediately as a new solutions is constructed instead of just after a few being generated.

Constraint Programming

Another AI technique used for resource scheduling problem is *Constraint Programming*. Constraint Programming is a programming paradigm based on expressing relations between variable in a form of constraints.

A Constraint Satisfaction Problem (CSP), a formal definition of a constraint problem, is characterized by a set of variable and for each variable a finite domain, as well as a set of constraints. The goal is to assign to every variable a value of its domain that does not violated any of the constraints.

The process of solving a CSP can be as simple as: select a variable, assign a possible value and test for violations of the constraints. If any violation is verified, another value is chosen. This process repeats until all variables are assigned. After each assignment it is possible to propagate to the remaining variables the effect of that assignment and quickly identify that certain path does not reach a consistent solution. This propagation step can be also done before any variable assignment.

Our problem may be described and solved as a CSP. However, previous work (Qu and He, 2009; Cipriano et al., 2006) showed that it may achieve good results for small instances of the problem but for large instances it can not produce results under decent time. For that reason, we do not try any solution with this technique.

2.1.3 Hybrid Approaches

The approaches presented in the Sections 2.1.1 and 2.1.2 can produce good results by themselves. Despite having good results, for large instances, they may have some performances problems. Combining approaches can mitigate or at least work around those issues.

As stated in Section 2.1.1, one way to deal with large instances, is by column generation. It is possible to replace the column generation component in branch-and-price to an equivalent CSP. Examples of this are (He and Qu, 2012; Trilling et al., 2006).

Constraint Programming can be very useful to local searches. It is more likely that a local search have a good performance if the initial point is good. A CSP can be used in order to obtain a solution and then the local search will improve it.

Despite being a promising approach, it may be an over worked technique for our resource scheduling problem.

2.2 Application Areas

To the best of our knowledge, there are a few published works on TSD alone (section 2.2.2). Other similar problems in the literature like *crew scheduling/rostering* applied to airlines and railways companies, as well as nurse roster scheduling contribute with helpful idea that can be used to solve TSD problems.

2.2.1 Educational Timetabling

Preparing an academic year at an university is a complex task. It is possible to identify five steps (I and Laporte, 1998). Teaching service distribution or Teacher assignment problem is one of those five steps of course scheduling. Sometimes the allocation of time, space and people is done at the same time, in a single step. This is called timetabling (Lewis, 2007). Examples of such solution is (Lü and Hao, 2010; de Grano et al., 2009; Schimmelpfeng and Helber, 2007).

As stated in Chapter 1, we cannot perform such tasks as a group. So, we will focus on analyzing the TSD subproblem from those that try to solve the timetabling problem.

Most of the work is based on modeling the timetabling problem as an assignment problem. An ILP model is defined and its main component is a decision variable X_{ijk} that take the value 1 when teacher i teaches class j on time k and 0 otherwise. Then, several constraints over variable X are also defined to address real restriction. (Schimmelpfeng and Helber, 2007; Al-Yakoob and Sherali, 2006)

It is clear that we can formulate our problem as an assignment problem.

There are also other methods for solving or at least build a good enough solution to these assignment problems. Timetabling problems can easily be converted into a graph coloring problem. In this problem, the goal is to assign one color to each vertex in such a way that no connected vertices have the same color. Nevertheless, this method fails when representing complex constraints (Lewis, 2007). It is not easy to describe everything in terms of vertices and edges of a graph.

Meta-heuristic methods (those presented in section 2.1.2) are widely studied on literature. Typically these methods are used in a two stage approach. The first stage is used to construct a feasible solution, i.e. a solution that at least do not violate any hard constraint. Then, in the second stage, the solution obtained is improved in order to avoid soft constraints violations. Genetic algorithms are also used to tackle this kind of problems.

A recent work on a similar problem (Phillips et al., 2015) used an ILP model to solve a class assignment problem (instead of assigning teacher, they assigned classrooms to class). The interesting part of this model is the fact that they proposed a model based on patterns. They proposed an exact method to achieve the best solution considering several measures in a sequentially process. At each step one criteria is optimized and then, in the next iteration the criteria from the previous iteration is added as an upper bound constraint.

2.2.2 Teacher Assignment Problem

Gunawan and Ng used an Indian University to propose an algorithm based on two local searches (Simulated Annealing and Tabu Search) and they showed that their proposed algorithm perform better than manual allocation and even genetic algorithms (Gunawan and Ng, 2011).

A very similar problem to ours was described in (Gunawan et al., 2008). The approach is based on a Genetic algorithm and it assumes that the faculty team is complete by the time the TSD is done. Their proposal was compared with the hand-made allocations and it is clear that the automatic one is better.

Both works incorporate domain specific knowledge to achieve their solutions. This problem tends to have a large dimension and generic methods may have a bad performance.

2.2.3 Staff Scheduling

In the transportation domain, the main goal, that is relevant to our work, is to have for each employee a monthly schedule that covers all the transportation work required, minimizes the cost for the company, and, at the same time maximizes the satisfaction of the workforce. A schedule/roster is a set of tasks - predetermined sequence of trips/flights, called pairings - and other individual activities. In order to create such schedules, there are two global approaches (Kohl and Karisch, 2004).

1. **Bidlines** - Anonymous schedules are first generated in order to minimize the cost for the company and then, employee rank/bid for the rosters they prefer. And finally the anonymous rosters are assigned in such a way that the satisfaction of employees is maximized.

2. **Personalized Rostering (also called cafeteria-style rostering)** - Employees express their preferences for certain attributes and then the rosters are generated for each employee considering those preferences.

In both cases the rosters must comply with certain rules and obligations, like each employee must achieve a certain number of flight credit (Gamache et al., 1998), days-off, minimum break time between pairings or qualifications/skills. Such systems, that helps to schedule employees taking into account preferences are called Preferential bidding Systems.

Crew rostering is very similar to our problem although our restrictions are slightly different.

Another similar problem is, in the health sector, to schedule nurses at health-care facilities. The most common formulation is: for each day, assign a shift to a nurse as well as assigning days-off. Typically this tasks is done for a 1-month period. The most common approach is to have predetermined shifts, such as, morning, evening and night shifts with fixed start and end times. Those shift periods coincide with the demand periods.

Both problems are similar with the one addressed in this thesis, however, both tend to be more complex because in all of them, there are some temporal continuity that do not make sense in our problem, such as hours, weekdays, days-off and rest periods between jobs. Spatial continuity is also important in crew scheduling but in TSD problems are not important.

Most of the approaches used for staff scheduling are based on the models presented in section 2.1.1. An ILP model is formulated and different techniques applied.

Column generation is widely used to deal with large problem instances. With this technique, domain-specific knowledge is incorporated. In crew scheduling, a common requirement is the advantage of senior employees over juniors. No junior employee can have their preferences satisfied if a senior one do not have his most preferable schedule.

At each step, the most senior employee without a schedule is picked up and the most preferable schedule is generated. It finishes when all employees have one schedule. All the schedules, which will be taken into account, are generated by solving a longest path problem with resource constraints and all the labour rules are modulated as resources in the problem, like minimum and maximum flight credits, consecutive working days, etc. Skills are implicitly encapsulated under the construction of the graph of the resource constrained longest path problem. If an arc exists it means that the employee can perform the task represented by that arc (Gamache et al., 1998).

A problem that arises is the fact that the most senior employee has always all of the preferred attributes in his schedule, harming the most junior one. A possible solution to this problem is to maintain all possible schedules for a senior employee that have the maximum score possible, instead of picking one and fix it. This ensures that any senior employee has the best possible score and allows the juniors to have it too (Achour et al., 2007).

Since the algorithm proposed by Gamache et al. is based on the seniority criteria, it may be difficult to apply on problems where it is not possible to order people by their seniority.

Achieving goals via objective function

Different goals and constraints lead to different approaches. In all those situations, we want to associate a resource to a job in such a way that there are not constraint's violations and the objective function is maximized (or minimized). As well as in the other problems, we want to ensure that all tasks will be associated to a resource. However it is not always true that all resources must be used. In our problem, this assumption may be true. All professor must have classes. Most of the objective functions are associated with productivity, such as minimizing the number of employees or machines for a certain number of jobs, maximized the number of tasks that resources can perform.

Both (Sabar et al., 2008; Eiselt and Marianov, 2008) focus primarily on the model formulation.

A complete model that addresses the staff scheduling problem for large assembly lines was proposed in (Sabar et al., 2008). There are several activities grouped in multiple workstations. Each activity has certain required skills which the assigned employee must have too. The model incorporates several time constraints that we will not detail. However, the objective function has a particular interest. In this model, the objective function is presented as a minimization of costs. Those costs are not only the normal ones, like cost for assigning an activity to a certain employee but also penalty costs for deviation from preferred characteristics for each employee.

No particular method is presented for this model as all the tested instances have a reduced sized. Despite the lack of testing on large instances, it is stated that it should not finish in reasonable time.

However, the results of this paper show another interesting point. The authors tested several objective functions and, by considering workers' preferences, the labor cost had a tiny variation but the schedules have incorporated the workers' preferences and consequently improve quality of life.

A similar objective function was also defined in (Eiselt and Marianov, 2008). Here, the objective function is defined as a sum of different costs in order to minimize the cost of overtime hours and contracted out tasks, minimize the distance between tasks' required skills and workers' skills and minimize the deviation of the actual workload for each employee from the average workload. Each task has certain required skills and those skills have levels. Each worker, according to their training, has certain skills at certain levels. So, it is possible to represent both tasks and workers as a point in the space where each dimension represent a skill and all their levels. With that, it is possible to define custom function to express the distance between points, in order to determine which worker is more capable of doing certain tasks.

An implicit goal programming model can also be used to solve a similar generic employee scheduling problem (Topaloglu and Ozkarahan, 2004). For every goal we want to achieve, there are two variables that indicate if there is a positive or a negative deviation from the target:

$$f(X) - (d^+ - d^-) = k$$

where $f(X)$ represent any function over the variables (X) of the problem, and d^+ and d^- are the deviation variables that indicate how much $f(X)$ is over or under k. These goals are added as a constraints and the objective function is a minimization of all the deviation variables that correspond to bad schedules. Weights associated with those deviation variables in the objective function are used to give more importance to some goals rather than other. Some organizations can established different importance for goals.

Looking at methods

The local search proposed by (Lü and Hao, 2012) was applied to nursing scheduling

A hybrid scatter search was proposed in (Maenhout and Vanhoucke, 2010) to solve a crew rostering problem. Another genetic algorithm was proposed to in order to solve the problem of a subway system. It was compared with other methods (tabu search and greedy algorithm) and it performed better for most of the instance of the problem (Elizondo et al., 2010). Identical solution was proposed to schedule doctors into emergency shift (Frey et al., 2009).

In (Cipriano et al., 2006), a similar problem to the nurse rostering problem was used to test an hybrid approach to solve those problems that combined CP and local search strategies. First a CP solver was used to create a feasible solution and then, this solution is given as the initial point to the local search strategies. They tested as well several local search strategies. The results showed that a Hill climbing strategy outperforms others strategies.

In another paper (Qu and He, 2009), the nurse rostering problem is divided into two phases, one to generate rosters via Constraint Programming (CP) and a second to improve those roster through

a variable neighborhood search strategy. This hybrid approach proved to be more effective than the complete CSP for large instances, it could not finish in time. It was also possible to verify that the usage of variable and value ordering heuristic for the resolution of the CP (phase 1) can dramatically improve the quality of the solution. In this paper, complete random selection of variables and values is useless upon a more informed heuristic. Results were compared with a genetic approach as well as another variable neighborhood approach. This approach produced better results in about 2/3 of the tests.

Chapter 3

Solution

Before we describe our system, we need to explain in detail what is the problem we address, which parameters we have to consider, and, which restrictions/constraints we have to take into account. The following sections describe the problem, the relevant data to the problem and how it is organized, as well as the explored techniques.

3.1 Problem Formulation

Our problem consists in assigning a *schedule*, that is a group of classes, to each professor according to some restrictions and guided by certain goals. In our approach, the classes that must be lectured are the set of tasks that must be accomplished and all the professors (employees) are the available resources. At IST, each professor is associated with two scientific areas (skills), each one representing a group of courses. Each course, according to the number of enrolled students must have a certain number of classes (course sections) that must be offered in each week because a classroom has limited seats. If a course has two course sections, this means that the enrolled students must only go to one of the course sections even though there are two possible course sections.

As an example, imagine that Course X has three hours of recitations and one and a half of lecture per week, if Course X has 150 enrolled students and a limit of 100 students per lecture and 30 students per recitation. Considering these numbers, Course X must have two course sections for lectures and five course sections for recitations in order to give the chance of all enrolled students to attend the classes needed. On the other hand, imagine that course X has 220 enrolled students. In this case, course X must have 8 course sections for recitations but having 3 course sections for lectures may lead to underused rooms. In this case, we may consider changing those limits of students per course sections, having in mind the idea that some repeaters students will not come to the classes.

In all the problems described in (Van den Bergh et al., 2013), there are temporal continuity constraints (tasks have a start and end time (one employee can only perform one task at each time), shifts also have start and end time). Our problem does not have temporal references. For the propose of this thesis, there are no collisions/overlapping between classes. Each class can be combined with all others.

As an effort metric, each professor indicates a number of *teaching credits* that he/she needs to achieve that academic year. Each class, based on its duration, semester length (number of weeks that it lasts), enrolled students, course's year are valued in terms of teaching credits. Therefore, the idea is that each teacher will be assigned to as many classes as his teaching credits allow.

The equilibrium must be achieved considering our secondary goals:

- Assign professors to classes within their scientific areas;
- Avoid part-time professors;
- Satisfy, as much as possible, the preferences of each professor.

There are restrictions that must be satisfied as well in order to consider a TSD as valid:

- All classes must be covered;
- The sum of the credits of the classes taught by each professor must not be less than the expected credits (plus/minus a threshold);
- All professors must have classes assigned.

For each academic year, professors must state their preferences but they can do it in the most varied way. In order to have a uniform preference's system, each preference represent a characteristic that must taken into account on the TSD. A professor can state their preferences about:

1. Semester
2. Campus
3. Course
4. Type of Class
5. Type of Class of a specific Course

The professor must also rate each choice with a preference from -10 (not desired) to $+10$ (highly desired). As an example, if a professor wants to teach practical classes from course X but not a single class from course Y, he/she must specify:

1. Practical Classes - Course X : 10
2. Course Y : -10

This model for preferences assumes that anything that is not stated has a neutral desire, as if it were stated with preference 0.

Another interesting point of this model is the fact that we can incorporate other preferences. Besides professors' preferences, the university itself is also able to state preferences (with higher rate values to give more relevance). IST, as in many universities, has their students filling a questionnaire at the end of each semester, for the professors' evaluation. In this way, the university can have feedback to

assess how good the professor is at that particular class. So, the university can state that professor A must teach classes from course X, but not from course Y, because the student's feedback indicates that professor A is really good on course X but have a not so good evaluation on course Y.

3.2 Data Models

In order to automate the problem described in Section 3.1, we must gather all the information that will be relevant in our solution.

We must distinguish two parts of the model: one is responsible for storing the information about professors and courses details (Generic Model); the other store only specific data about the classes and professors that is dependent of the academic year (Specific Model).

Figure 3.1 shows a UML representation of both data models that were detailed in the following subsections.

3.2.1 Generic Model

A professor has some identification data, a rank and two scientific areas that indicate the courses that he/she is able to teach. In some cases, it is possible to teach courses outside those areas. We will not explore the attribute Rank nor the ability to teach courses from other scientific areas beyond their two scientific areas.

- Professors:
  - ID
  - Name
  - Rank
  - Main Scientific Area
  - Secondary Scientific Area
  - Other Courses (outside Main and Secondary Scientific Areas)

A course belongs to a scientific area, it has a location (IST is divided into three campi) and a semester. It is also associated with a certain course year (degrees are divided into years, and each course year has a set of courses that students have to attend), as well as a teaching workload.

- Course:
  - ID
  - Name
  - Scientific Area
  - Campus

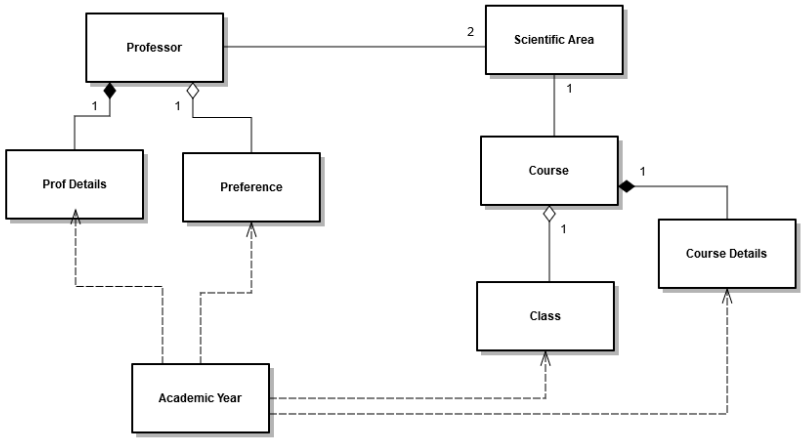

Figure 3.1: UML representing the both data models

- Semester
- Course Year
- Teaching workload

3.2.2 Specific Model

Concerning the professors' information that varies each academic year, it is only necessary to store their expected credits.

- Professor (Prof Details in Figure 3.1):
  - Professors' ID
  - Academic Year
  - Expected Credits

The model also needs to represent estimates of the enrolled students and, consequently, how many credits a course will value. It is important to have a forecast of enrolled students. Dividing the forecast between first attendees and repeaters allows to better estimate the number of course sections as repeaters tend to go less to classes.

- Course (Course Details in Figure 3.1):
  - Course ID
  - Academic Year
  - Enrolled Students - First Attempt (estimate)
  - Enrolled Students - Repeaters (estimate)
  - Estimated Credits Value

Each class has a type (lecture, recitation, seminars), an hour duration, and a duration in terms of weeks in the semester. There are classes that do not last the entire semester.

- Class:
  - ID
  - Course
  - Academic Year
  - Type
  - Duration
  - Semester Duration (in weeks)

A preference is associated with a professor. It has a type, an attribute that indicates the real desire like which campus (A or T) or semester (1 or 2) or even both, as well as the rate.

- Preferences:
  - Professor
  - Academic Year
  - Type
  - Attribute
  - Rate

3.3 Techniques Explored

In this Section, we provide a description of the techniques used in our work. We explored techniques from: Operational Research and Artificial Intelligence.

3.3.1 Operational Research - Mathematical model

From the literature review, it was clear that there are two different models that can be applied. One is based on resource assignment, where we directly assign a resource to a group of tasks. The other is based on schedule assignment, where we assign resources to schedules.

In the following sections, we describe the two models tested for our problem.

Resource Assignment

One suitable model for our problem is based on the first model presented in Section 2.1.1. The basic idea is to directly assign each class of a certain type of a certain course to each professor.

Let us assume that we have a decision variable X_{ctp} which contains a positive integer representing the number of classes of type t of course c that the professor p will teach.

$$X_{ctp} \in \mathbb{Z}_0^+ \tag{3.1}$$

Let us also assume a set of auxiliary parameters:

- P_{ctp} - Preferences value of classes of type t of course c to the professor p;
- V_{ct} - Teaching credits value of classes of type t of course c;
- M_p - Teaching credits of professor p;
- Bl and Bu - Lower and upper bounds for the teaching credits;
- LC_{ct} and UC_{ct} - Minimum and maximum number of times that the class type t of course c has to be taught, respectively;
- H_{ctp} - Qualification of professor p to teach classes of type t of course c;
- ω_t - Set of types of classes;

If we denote by Tp the total number of professors and Tc the total number of courses, expressions 3.2 to 3.5 represent all the constraints we need to have.

$$M_p - Bl \leq \sum_{t \in \omega_t} \sum_{c}^{Tc} V_{ct} X_{ctp} \leq M_p + Bu \qquad p = 0, 1, 2, 3, .., Tp \tag{3.2}$$

$$\sum_{p}^{Tp} X_{ctp} \in [LC_{ct}, UC_{ct}] \qquad \forall_{ct} \tag{3.3}$$

$$X_{ctp} \geq 1 \implies H_{ctp} = 1 \qquad \forall_{ctp} \tag{3.4}$$

$$\tag{3.5}$$

The only piece missing is the objective function. In this case we need to maximize the preferences for each professor:

$$\sum_{p}^{Tp} \sum_{t \in \omega_t} \sum_{c}^{Tc} P_{ctp} X_{ctp} \tag{3.6}$$

This model has some limitations: a) It is not possible to have restrictions over the number of courses a professor will teach; b) It is not possible to impose bounds for the number of courses a professor may teach. To solve this problem, we introduced a binary decision variable C_{cp} that has the value 1 if professor p teaches at least one class of course c, and a constraint responsible for updating this decision variable ($C_{cp} \in \{0, 1\}$):

$$\sum_{t \in \omega_t} X_{ctp} \geq 1 \implies C_{cp} = 1 \qquad p = 0, 1, ..., Tp \quad c = 0, 1, 2, ..., Tc \tag{3.7}$$

With this variable, we can constraint the number of courses a single professor can teach by adding the restriction:

$$\sum_{c}^{Tc} C_{cp} \leq k \qquad p = 0, 1, 2, 3, .., Tp \tag{3.8}$$

This ensures that no professors teaches more than k different courses, even though they can teach more than one class in each course.

The model presented earlier represents the basic model for our problem. In order to improve and enrich our model we had another objective: minimize the sum of all differences between the real value of the assigned classes and the desired teaching credits (Equation 3.9). Another equivalent objective is to minimize the average differences between the real value of the assigned classes and the desired teaching credits (Equation 3.10).

$$\sum_{p}^{T_p} \left| M_p - \sum_{t \in \omega_t} \sum_{c}^{Tc} V_{ct} X_{ctp} \right| \tag{3.9}$$

$$\frac{\sum_{p}^{T_p} \left| M_p - \sum_{t \in \omega_t} \sum_{c}^{Tc} V_{ct} X_{ctp} \right|}{T_p} \tag{3.10}$$

Even tough, ILP do not allow multiple objectives in a straightforward way, there are at least two methods that allow to have multiple objectives. The first method is the composition of multiple objectives in a single expression such as:

$$Obj(X) = \alpha_1 O_1 + \alpha_2 O_2 \tag{3.11}$$

where O_1 and O_2 are the individual objectives and α is a coefficient that represents the relative importance of the objectives.

The second method consists on sequentially optimizing with a single objective at each time. We start with the basic model, solve it, and store the objective value (S_i) of the solution that was found. Then, we add a constraints to the basic model:

$$O_i = S_i \tag{3.12}$$

and solve it again. If there are more objectives to be optimized in the next iteration, we repeat the process, but considering the model with all constraints (3.12) previously added.

These two methods for optimizing multiple objectives originated two different models where the two objectives expresses in Expression 3.9 and 3.10 are used as a weighted objective function (Expression 3.11) or as a multi-objective sequential optimization problem.

(Pre-generated) schedules assignment

The schedules assignment model assumes that we first generate all possible schedules (combination of classes) for each professor or at least generate a very large number of them.

In order to generate the schedules, we consider all the combinations between all the classes a

professor can teach, that are inside the interval of expected credits. We could use all possible schedules, however we applied heuristics to decrease the number of schedules per professor, such as eliminate all schedules that have negative or zero rating assuming that the professor has stated at least one preference.

Then, for the model definition, we have a decision variable X_{sp} that takes the value 1 if the schedule s for professor p is chosen ($X_{sp} \in \{0, 1\}$).

Some additional parameters need to be set to correctly define the constraints:

- A_{scp} that indicates how many classes c are present on schedule s for professor p.
- LC_c and UC_c - Minimum and maximum number of times that class c has to be taught, respectively.
- P_{sp} - Preferences value of schedule s to the professor p

Let denote by Tp the total number of professors, Tcc the total number of classes and Ts the total number of schedules.

In this model, we have to ensure that the selected schedules comply with the minimum and maximum number of times professors teach a certain class.

$$\sum_{p}^{Tp} \sum_{s}^{Ts} A_{scp} X_{sp} \in [LC_c, UC_c] \qquad c = 0, 1, ...Tcc \tag{3.13}$$

Another important aspect is to ensure that only one schedule is selected by each professor.

$$\sum_{s}^{Ts} X_{sp} = 1 \qquad p = 0, 1, 2, .., Tp \tag{3.14}$$

The objective is still to maximize the preferences values of the selected schedules.

$$\sum_{p}^{Tp} \sum_{s}^{Ts} P_{sp} X_{sp} \tag{3.15}$$

Constraints such as the maximum number of courses per professor and others are incorporated in the schedule generation process. We only generate valid schedules.

3.3.2 Artificial Intelligence - Local Search

AI also gives contributions to our problem. We have to define three different aspects concerning local searches: *i*) how to get the initial solution. *ii*) what moves we consider *iii*) which search strategies to use

Local search only works for complete solutions even though the starting point may be an invalid solution. So, a fundamental step is to construct the initial solution. For this step, previous work showed that a better initial solution produce a better final solution. We built a constructive heuristic in order to obtain a reasonable solution and then the local search tries improving it. This heuristic sequentially tries to assign each class to the professor who has the highest sum of preferences rating that involve that class. In case of more than one professor with the same sum or if no professor stated a preference

about that particular class, we decide randomly which professor will teach the class. This is a greedy approach.

In order to develop a local search, we had to define the moves that we will use. Based on the literature review, we consider a move to be: a *swap* of classes between 2 professors, and a *direct move* of a class from one professor to another.

Concerning the strategies, we decided to use Hill Climbing and Tabu Search. These strategies combined with our moves can produce a large number of changes in a short period of time. This fact can help on achieving a better solution, i.e. improving the initial solution. In both strategies the objective function is the weighted sum of the preferences value and the total deviation of credits. The Tabu list used in the Tabu Search can have single moves or whole states and the size will determine the memory aspect of this search strategy.

Chapter 4

Architecture

In this chapter, we present the details about the architecture of our solution. Our solution was conceived to be modular, having in perspective future uses of it. One fundamental aspect of our system is versatility as it allows testing multiple and equivalent components that help to improve our solution and build a more valuable system.

IST uses FenixEdu[1] as their school management system. One alternative was to develop the system right away integrated with FenixEdu. However, our solution only looks into DEI, and IST has several other departments that may have a different process of assigning professors to courses. Another alternative was to build a solution as a standalone platform but considering future integration in FenixEdu. We chose this last alternative.

4.1 General Overview

Our system is divided into three main independent components as in shown Figure 4.1:

- Web Platform - used by professors and department's staff to introduce data;
- Database - to store all data in a persistent way;
- TeachOpt Solver - responsible for the automatic step of generating a TSD.

The arrows in Figure 4.1 represent data flows between the components. The interactions with the database are read/write actions, but the interaction between the web platform and the software to generate the solution is just an activation interaction. In this case, the platform starts the software; however, the result is written in the database and not directly returned to the web platform.

In the following sections, we present the details for the web platform as well as for the software responsible for generating the TSD. The database component is used just to store data. There is no logic in this component.

[1] http://fenixedu.org/

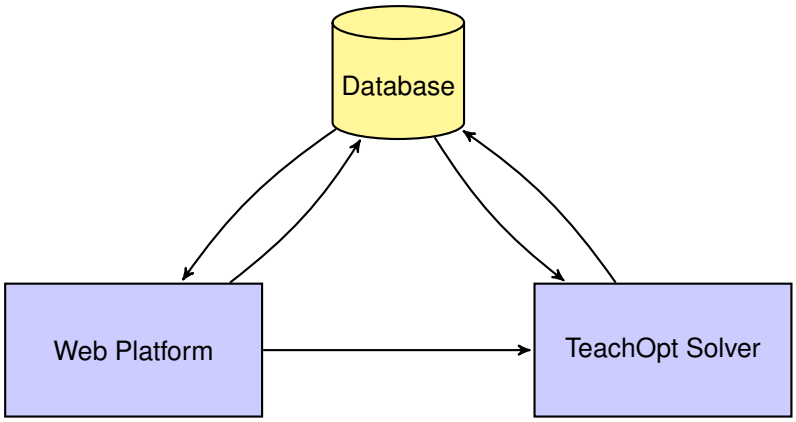

Figure 4.1: TeachOpt architecture schema

4.2 Web Platform

In order to automate the process of generating a TSD, a web-based platform was built. We decided to use a web interface because it allows users with a Internet connection to be autonomous and insert his/her preferences and view the generated TSD.

In the web-platform, we separated the features into areas according to the user that accesses it. Users are divided into two different user profiles:

1. Professor - each professor has an account with his/her user profile.
2. Department - every person in the department access and modify the information as well as plan the TSD. This is an administration profile.

Each user profile accesses separate areas of the platform and different features.

- Professor
    1. Edit personal information;
    2. Insert, change and remove preferences for each TSD process;
    3. View the complete generated TSD.
- Department
    1. Insert, change and remove professors, courses, scientific areas, workload of each course;
    2. Create TSD process;
    3. Insert preferences on behalf of a professor;
    4. Insert the department preferences;
    5. Start the TSD process;
    6. View the generated TSD.

Figure 4.2 is a screenshot of one screen of the web platform. There is a static menu on the left side of the screen where the user can find every option available to him/her. The content is presented in

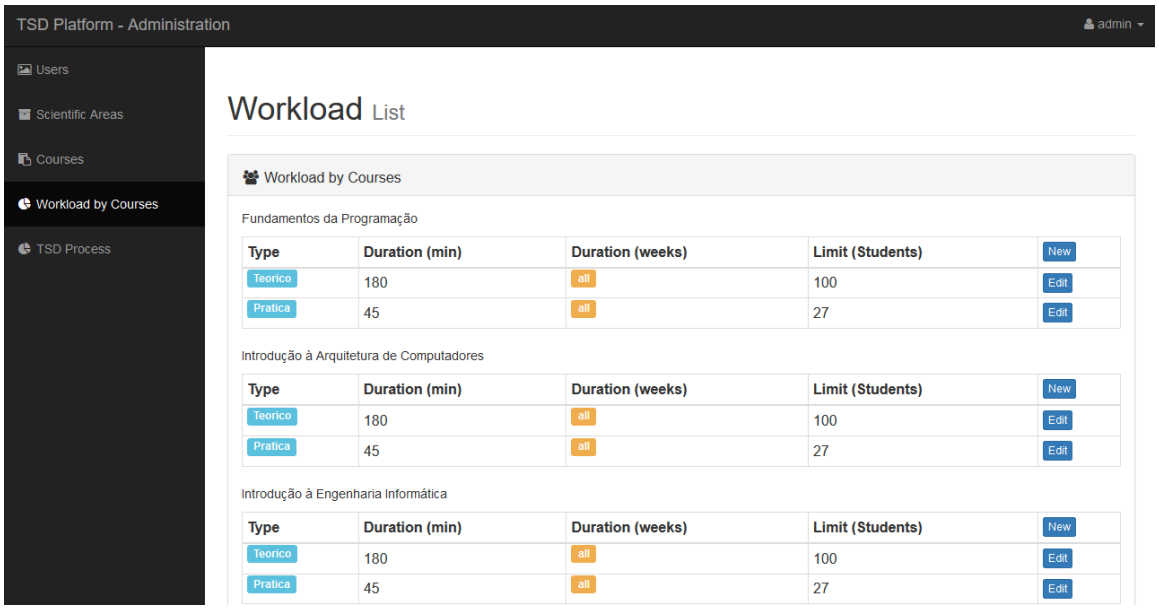

Figure 4.2: Web Plaform Screenshoot

the white area. In the case of Figure 4.2, it is the course's workload management feature, where we can see, for instance, that the first course that appear (Fundamentos da Programação) has two types of classes, one called Teórico (Lecture) and another called Prático (Recitation). In the first type, there are two classes a week with a duration of 90 minutes making a total of 180 minutes per week, they last all semester and have a limit of 100 students per session. In the second type, the classes are 45 minutes long, they last all semester and have a limit of 27 students per session.

4.3 Database

The database component does not have any special feature or function. It is used to store data in a persistent and consistent way. Both the web platform and the TeachOpt solver connect to it to read information and store other information.

4.4 TeachOpt Solver

Concerning the automatic step of this process, we created a software that can generate the TSD. This software makes a bridge between the database and the solver that is capable of solving OR problems or the AI solvers that use local searches.

In order to generate a TSD, it starts by reading the database and create an intermediate structure. This intermediate structure helps to avoid dependency of the solver with the specificity of the database. Then, the solver is initiated and the model is configured based on the formulas specified in Section 3.3.1. Finally, the variables/parameters are filled according to the intermediate structure.

The interface of the software (Figure 4.3) also allows to configure some parameters about the solver and to view the generated TSD before it is save into the database.

There are two parameters that affect the problem itself. The Lower Bound and the Upper Bound (that appear in the left side of Figure 4.3) are used to determine the interval where the teaching credits of each professor must be, taking into account the desired teaching credits stated by each one. The higher the values are, the more easily a solution is found. However, it may be a poor solution as the difference between desired and effective credits can be big. The other two parameters that appear in the middle of the image are used to know which TSD process must be loaded and what will be the name of the solution to that process that is used to identify the solution on the web platform.

It is important to note that if no feasible solution is obtained, i.e. if no solutions that respect all the constraints is found, one of the following error messages will appear:

- UpperBound - Credits
- LowerBound - Credits
- LowerBound - Classes
- UpperBound - Classes

The first two messages appear when, for at least one professor, the credits constraints defined in Equation 3.2 are violated. The other two appear when the constraint about the number of classes of a course is not right (Eq. 3.4).

Conceptually these four messages represent one of the following problems:

1. Not enough professors to teach all the classes;
2. All classes are assigned, but some professors have a low level of credits.

If one of these conditions is true, there are 3 possible ways to correct them. One is to increase (or decrease) the number of classes. Another, that only applies to condition 1, is to hire part-time professors. The last one is to relax the credits boundary constraint. Only this last one can affect the model; all the others only affect the data itself.

We do not have any specific mechanism on the OR models to address the problem of no feasible solution being found.

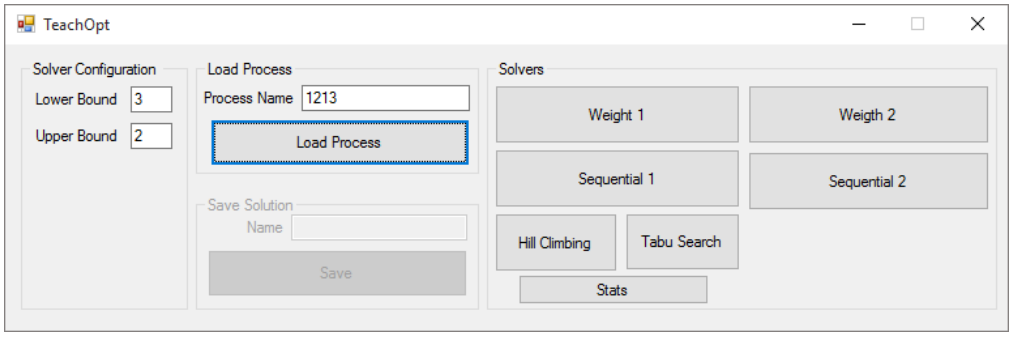

Figure 4.3: TeachOpt Solver Screenshot

4.5 Overview of the process

Bellow, we show how the TSD is done, in our system:

1. Insertion of data related to academic year, such as (not) available courses, forecasts of enrolled students and number of lectures and recitations per course according to their workload;
2. Professors will be asked to introduce their data and preferences;
3. Department will generate the TSD;
4. Professors will be notified about their assignments as well as if their expected credits is reached.

Chapter 5

Implementation

To implement the features discussed in the Section 4, we made some decisions about the implementation and tools used. The usage of well-known tools allowed us to avoid simplifications of the architecture. As we do not explore any special feature from the adopted tools, all parameters of all tools were set to the default values.

The following sections describe the choices made about the implementation of our system.

5.1 Web Platform

5.1.1 Simplifications

The simplification that was made to our proposed solution was the ability of the web platform to start TeachOpt Solver. We did not implemented this feature due to issues with the infrastructure where the solution runs. Therefore, the TeachOpt Solver must be started by running its executable. However, this simplification does not affect the purpose of our solution.

Every other feature described in Section 3 was implemented and is fully functional in the web platform.

5.1.2 Technology

To develop the web platform, we used PHP. There were alternatives, but this choice was made because it is a simple programming language that allows easy development of websites. Our previous knowledge about it, as well as easy access to infrastructure that support it, contributed for our choice.

Concerning the look & feel and to avoid having a plain HTML site, we used Bootstrap [1] to give a nicer look to the web pages. It is a ready to used library that, with no effort, can transform the web pages into something more appealing to the users.

[1] http://getbootstrap.com/

5.2 Database

The database where all the data is stored is the one of the key pieces of the system. It is the bridge between the user interface and the module that is responsible for the automatic generation of the TSD, TeachOpt Solver.

It is important to note that we just need to use the basic operation of databases to access and manipulate the data: Select, Insert, Update and Delete.

5.2.1 Model

We chose to have a relational model, rather than other possibilities, because it is widely used as it allows to have the model presented in Section 3.2 with minor changes. It is out of the scope of this thesis to explore new methods to store information. We only want any easy and proven method to store it.

5.2.2 Database Management System

There are plenty of DBMS that implement a relational model. Despite having different features, all of those DBMS have a common base. We are just interested in those base features. So our choice of DBMS was based again in easy access to infrastructure. We chose MySQL[2].

Switch from this DBMS to another will not have any performance implications. There are no complex queries that may take advantages from better query optimizers. All queries are basic with no advanced operators.

5.3 TeachOpt Solver

5.3.1 Generic Behavior

TeachOpt Solver is the core software of our system. In order to easily test different techniques, we implemented a two-step process: *1*) read the DB data into the intermediate structure; *2*) start the solver using the intermediate structure. The first step is done by a single DB reader, as we do not have multiple sources of information. The second step is where the TSD solution is created (solved). As stated in Section 3.3, we needed to test several techniques, so we created multiple solvers. These solvers have the same structure and interface but implement different techniques.

5.3.2 Solvers

Solvers represent one of the major pieces of the system. They are responsible to generate the TSD solution.

In the OR methods, it is necessary to solve all the mathematic models presented in Section 3.3.1. Implementing the algorithms that solve those models is a complex and hard task. Several software

[2]https://www.mysql.com/

products already have implemented those algorithms. In most of them, it is only necessary to specify the models and data and they can produce the solution without further work.

Since in OR methods there are the independence between model and solution technique, we can just pick one of solver from those that allows to solve ILPs. We selected CPLEX[3] from IBM.

CPLEX is a widely used tool in the industry and very powerful. Despite being a very expensive tool, it is available to research, for free, via IBM Academic Initiative[4].

To program and use this solver, we used the API available to C# in order to create the models described in Section 3.3.1. The other possibility was to write a file in a specific syntax that CPLEX could read. This option was not considered as it had an overhead of learning the new syntax.

The AI solvers were completely implemented by us. We did not used any external library in order to have more control over the solution generation process. Every aspects of the AI solvers were implemented using the basic data structures and built-in features of C#.

[3] IBM software property - http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[4] http://www-304.ibm.com/ibm/university/academic/pub/page/academic_initiative

Chapter 6

Evaluation

To assess the quality of any work, it is important to perform evaluation tests and take conclusion according to the results. Our problem, by itself, is not very explored in the literature, so it is not easy to compare results. Another aspect that makes a comparison difficult is that each problem has its own specification.

In this chapter, we evaluate and discuss our solution. We analyze the quality and acceptance of the generated solutions. Then, we explain why some techniques clearly failed to produce good results or to finish in reasonable time. Finally, we discuss how easy or hard it is to add features to our solution.

6.1 Evaluation Sets and Environment

In order to perform the tests we created several small datasets as well as datasets based on real TSDs. The small datasets consist on 3 professors and 5 courses, but with different teaching credits as well as enrolled students.

The small datasets were used to manually check if the solution produced by our system respected all constraints. For each small dataset, we generated 3 solutions. Each solution was manually checked in order to guarantee that all teaching credits boundaries for all professors were respected; if the calculation of course sections is correct; if the courses assigned to each professor are from their scientific areas.

From all solution generated and then manually checked, we did not find any constraint violation. These results ensured that our model is correct that it does what is expected.

The datasets based on real TSDs have 18 professors and 22 courses representing one from five scientific areas existing in DEI. These datasets based on real data had preferences that we manually extracted from the TSD. Some teachers had generic preferences, and others had more specific preferences.

The following table details the two datasets based on real TSDs:

	Number of professors	Number of courses	Average desired credits	Minimum desired credits	Maximum desired credits	Total desired credits
Dataset 1	17	22	10.25	4.5	16	174.25
Dataset 2	18	22	10.19	4.5	18	183.5

Table 6.1: Description of the dataset based on real TSDs

All tests were run in a computer with an Intel Core i7-4702MQ cpu and 16gb of RAM. The IBM CPLEX version used was v12.6.1.

6.2 Solution quality

OR techniques (Resource Assignment - Section 3.3.1) were exhaustively tested to assess the real quality of the produced solution. We first had to ensure that the created model respect all the problem constraints, then compared the generated solution with the manually generated solution, and, for last, checked with an experienced person if the solution could be applied.

6.2.1 Comparison with real solutions

In Table 6.2, we have a description of what dataset and technique was used to produce each solution. We have one solution of each combination of dataset with resource schedule model as presented in Section 3.3.1. The first two columns indicate which of the datasets described in Table 6.1 were used in that solution. The following column indicates which objectives were used. The last two columns indicate which of the methods described in Section 3.3.1 were used.

	Datasets		Objectives			Optimization Methods	
	Dataset 1	Dataset 2	Maximize preferences	Minimize total deviation of credits	Minimize average deviation of credits	Weighted $(\alpha_1 = 1, \alpha_2 = -10)$	Sequential
Solution 1	X		X	X		X	
Solution 2	X		X		X	X	
Solution 3	X		X	X			X
Solution 4	X		X		X		X
Solution 5	X		Manual				
Solution 6		X	X	X		X	
Solution 7		X	X		X	X	
Solution 8		X	X	X			X
Solution 9		X	X		X		X
Solution 10		X	Manual				

Table 6.2: Solution Details

Comparing the solution obtained by our approach with real TSD allows to determine if our solution is conceptually correct and produce satisfactory results. The comparison between some key indicators of the solutions are shown in Table 6.3.

Solutions 5 and 10 represent the solutions produced by humans. In the first column of Table 6.3, we have the solution value based on the used preferences calculated by summing the value of all preferences that were satisfied in the solution. Having Solution 5 as the baseline for the first four solutions, we

	Preferences Value	Total Deviation from Expected Teaching Credits	Average Deviation from Expected Teaching Credits	Exec. time (s)
Solution 1	1180	23.75	1.39	< 1
Solution 2	1200	26.75	1.57	< 1
Solution 3	1200	26.75	1.57	< 1
Solution 4	1200	26.75	1.57	< 1
Solution 5	1300	24.5	1.44	< 1
Solution 6	2550	14.5	0.81	< 1
Solution 7	2720	28	1.56	< 1
Solution 8	2780	34	1.89	< 1
Solution 9	2720	28	1.55	< 1
Solution 10	2680	40.8	2.27	< 1

Table 6.3: Preferences and Expected Credits measures

can see that we have a 10% decrease of the solution value from the manually generated solution. For the other four solutions which have the solution 10 as baseline, we have a decrease of 5% on solution 6 but we also have an increase of about 2% on solutions 7-9.

Considering the other two measures presented in Table 6.3, we must take them into account to assess the real quality of the solutions. According to these two measures, Solution 1 and Solution 6 are better than their corresponding baseline solutions. This means that, in total, there are less credits over or under the expected. Also note that the difference between solution 6 and 10 is very significant.

6.2.2 Manual Validation

We asked the person who performs the task of elaborating the TSD for several years, to informally evaluate each of the 10 solutions. The evaluation consisted on using her experience to assess the correction of the solution concerning the common sense and if it was possible to have that TSD implemented.

The main defects of the solutions were about some mismatches between professors and courses even if it were within his scientific areas. Other comments were about having some professors assigned to just one single lecture of a course.

Both the comments have the some cause. As one of the objectives is to minimize the difference between expected teaching credits and real teaching credits, the system tries to fill all gaps with other classes that does not affect the solution value. To fill those gaps, sometimes a class outside his/her preferences is allocated and it leads to the problem described in the comments. We suggest a solution to these problems on Section 7.3.

6.3 Why some techniques fail in our problem?

During the implementation of local searches, we realized that, for our problem, they do not produce good results. One major problems is the quality of the initial solution. Starting from a better solution allows the search to achieve better results. However, one of the key requirements of the initial solution is that it must be a valid solution. Constructing such solution is not an easy task and most of the time, our greedy algorithm could not do it.

Another problem with this technique is the amount of successors that can be generated at each step

of the search. It generates one successor for each professor and for each course section of a course. For instance, for dataset 1, we have 17 professors and $22*2$ course sections (assuming that each course has two course sections), so it generates $17*22*2 = 748$. Only just a small percentage of the successors correspond to a valid solution.

In fact, our problem is not suitable for this technique. The amount of computation required to ensure that all constraints are always respected is huge comparing with the OR techniques used.

Another technique that revealed to be inefficient is the Schedule Assignment model (Section 3.3.1). To use this model, we first need to generate a large number of schedules. This task is very complex and must be done for each professor individually due to the fact that each professor has his/her own teaching credit limit. The number of schedules that were generated is unpractical as well as the time it takes to generate them. Having a huge number of schedules affects the OR models as it has to deal with each single one. This fact also affects performance.

Local searches or the Schedule Assignment model take considerable more time than the Resource Assignment model evaluated in this chapter to obtain the some solution. The Resource Assignment model retrieves a solution almost instantaneously with guaranteed optimal solution for our model.

6.4 Extensibility of the solution

Another important point of our solution, web platform and software generator of TSD, is the fact that this is generic enough to allow to be extended.

If we want to add a new type of preference to the system, we need to add a table in the database; then add the form to the web platform in order to be able for the professor to specify this new type of preference; finally, we need to add the computation of that type of preference to the internal representation of the software to automatically generate the TSD.

If we need to change the method of how the course sections is calculated, we can do it directly on the web platform as it allows to change same parameters that affect this calculation, or if it is a more extreme change, we must do it in the implementation of the software to automatic generate the TSD.

IST uses FenixEdu as their school management system. Our platform is a standalone application. All data that is needed can be added via web platform. But, if, in the future, we start to use the FenixEdu database, we need to indicate how to map their database into the internal representation. The TSDs will be produced like with our web platform.

Chapter 7

Conclusions

7.1 Summary

Assigning professors to courses is not an easy task and done under very different approaches in the schools that do it. We analyzed how DEI performs such task in order to conceive a system that automates it.

Our literature review showed that there are two main areas that contribute to solve this problem: OR and AI. We developed models for both approaches. Two models for OR were created. One based on assigning directly courses to professor and the second based on assigning a pre-generated schedule to a professor. We also used two search strategies from AI: Hill Climbing and Tabu Search. Both strategies use as basic moves: changing a class from one professor to another and a swap of classes from two professors, one class from each.

From our tests done using datasets that are based on real TSD, the OR model that makes the direct association between courses and professors outperforms any other technique.

7.2 Achievements

Our work allowed to develop a solution that can be used by DEI in order to start automating the process of assigning professors to courses and classes. Our system can generate good enough solutions that are valid, that can satisfy the same number of preferences of a manual solution and decrease the deviation of the difference between expected and effective credits. Nevertheless, sometimes the solutions may need some small modifications or improvements that must be done manually.

7.3 Future Work

Our work can have improvements to get better results. One of the problems that raised is how the number of courses sections is calculated. We used a basic rule to do it. However sometimes it does not

get it right. Using the knowledge from previous year, as well as using data from students attendance to classes, can be very useful to have better predictions of number of the classes needed.

Another interesting point to improve is the objective function of the OR models. Adding other objectives that make sense or even giving the possibility to the user to choose which of them he wants to use may help to avoid the problems pointed in the manual validation.

Bibliography

Achour, H., Gamache, M., Soumis, F., and Desaulniers, G. (2007). An Exact Solution Approach for the Preferential Bidding System Problem in the Airline Industry. *Transp. Sci.*, 41(3):354–365.

Al-Yakoob, S. M. and Sherali, H. D. (2006). Mathematical programming models and algorithms for a class-faculty assignment problem. *Eur. J. Oper. Res.*, 173:488–507.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-Price: Column Generation for solving huge integer programs. *Oper. Res.*, 65(3):316–329.

Cipriano, R., Gaspero, L. D., and Dovier, A. (2006). Hybrid approaches for rostering: A case study in the integration of constraint programming and local search. In *Hybrid Metaheuristics*, volume 4030, pages 110–123. Springer Berlin Heidelberg.

Dantzig, G. B. (1964). *Linear Programming and Extensions*, volume 34. Princeton university press.

de Grano, M. L., Medeiros, D. J., and Eitel, D. (2009). Accommodating individual preferences in nurse scheduling via auctions and optimization. *Health Care Manag. Sci.*, 12:228–242.

Eiselt, H. a. and Marianov, V. (2008). Employee positioning and workload allocation. *Comput. Oper. Res.*, 35(2):513–524.

Elizondo, R., Parada, V., Pradenas, L., and Artigues, C. (2010). An evolutionary and constructive approach to a crew scheduling problem in underground passenger transport. *J. Heuristics*, 16(4):575–591.

Frey, L., Hanne, T., and Dornberger, R. (2009). Optimizing staff rosters for emergency shifts for doctors. In *2009 IEEE Congr. Evol. Comput. CEC 2009*, pages 2540–2546.

Gamache, M., Soumis, F., Villeneuve, D., Desrosiers, J., and Gelinas, E. (1998). The preferential bidding system at Air Canada. *Transp. Sci.*, 32(3):246–255.

Glover, F. (1989). Tabu Search - Part I. *ORSA J. Comput.*, 1(3):190–206.

Gunawan, A. and Ng, K. M. (2011). Solving the teacher assignment problem by two metaheuristics. *Int. J. Inf. Manag. Sci.*, 22:73–86.

Gunawan, A., Ng, K. M., and Ong, H. L. (2008). A genetic algorithm for the teacher assignment problem for a University in Indonesia. *Int. J. Inf. Manag. Sci.*, 19(1):1–16.

He, F. and Qu, R. (2012). A constraint programming based column generation approach to nurse rostering problems. *Comput. Oper. Res.*, 39(12):3331–3343.

I, M. W. C. and Laporte, G. (1998). Recent Developments in Practical Course Timetabling. *Pract. Theory Autom. Timetabling II*, 1408:3–19.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science (80-.).*, 220(4598):pp. 671–680.

Kohl, N. and Karisch, S. E. (2004). Airline Crew Rostering : Problem Types , Modeling , and Optimization. *Ann. Oper. Res.*, 127(1-4):223–257.

Lawler, E. L. and Wood, D. E. (1966). Branch-And-Bound Methods: A Survey. *Oper. Res.*, 14:699–719.

Lewis, R. (2007). A survey of metaheuristic-based techniques for University Timetabling problems. *OR Spectr.*, 30(1):167–190.

Lü, Z. and Hao, J. K. (2010). Adaptive Tabu Search for course timetabling. *Eur. J. Oper. Res.*, 200(1):235–244.

Lü, Z. and Hao, J.-K. (2012). Adaptive neighborhood search for nurse rostering. *Eur. J. Oper. Res.*, 218(3):865–876.

Maenhout, B. and Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *Eur. J. Oper. Res.*, 206(1):155–167.

Phillips, A. E., Waterer, H., Ehrgott, M., and Ryan, D. M. (2015). Integer programming methods for large-scale practical classroom assignment problems. *Comput. Oper. Res.*, 53:42–53.

Qu, R. and He, F. (2009). A hybrid constraint programming approach for nurse rostering problems. In *Appl. Innov. Intell. Syst. XVI - Proc. AI 2008, 28th SGAI Int. Conf. Innov. Tech. Appl. Artif. Intell.*, pages 211–224. Springer London.

Sabar, M., Montreuil, B., and Frayret, J.-M. (2008). Competency and preference based personnel scheduling in large assembly lines. *Int. J. Comput. Integr. Manuf.*, 21(4):468–479.

Schimmelpfeng, K. and Helber, S. (2007). Application of a real-world university-course timetabling model solved by integer programming. *OR Spectr.*, 29:783–803.

Topaloglu, S. and Ozkarahan, I. (2004). An Implicit Goal Programming Model for the Tour Scheduling Problem Considering the Employee Work Preferences. *Ann. Oper. Res.*, 128:135–158.

Trilling, L., Guinet, A., and Le Magny, D. (2006). Nure scheduling using integer linear programming and constraint programming. *INCOM- 14th IFAC Symp. Inf. Control Probl. Manuf.*, pages 671–676.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., and De Boeck, L. (2013). Personnel scheduling: A literature review. *Eur. J. Oper. Res.*, 226(3):367–385.

Whitley, D. (1994). *A Genetic Algorithm Tutorial*, volume 4. Kluwer Academic Publishers.

Williams, H. P. (1980). Integer programming. *Constraints*, 11 Suppl 1(-1):272–319.