Escuela Politécnica Superior

Dpto. de Ingeniería Informática

Doctorado en Ingeniería Informática y de Telecomunicación

Doctoral Thesis

New Swarm-Based Metaheuristics for Resource Allocation and Scheduling Problems

Author

ANA MARIA NOGAREDA VILARIÑO

Advisor

Dr. D. DAVID CAMACHO FERNÁNDEZ

Thesis submitted for the degree of:

Doctor of Philosophy

July 2017

Department: Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid (UAM)
SPAIN

PhD Thesis: "New Swarm-Based Metaheuristics for Resource Allocation and Scheduling Problems"

Author: **Ana María Nogareda Vilariño**
Ing. Math. Dipl. EPFL
Ecole Polytechnique Fédérale de Lausanne, Switzerland

Advisor: **David Camacho Fernández**
Doctor Ingeniero en Informática
(Universidad Autónoma de Madrid)
Universidad Autónoma de Madrid, SPAIN

Year: 2017

Committee: President:

Secretary:

Vocal 1:

Vocal 2:

Vocal 3:

Dedicado a Mateo y a David,

porque mamá siempre tenía muchos deberes,

y a Vincent,

sin él, este trabajo no hubiera sido posible.

"La nature hait l'uniformité et aime la diversité. C'est là peut-être que se reconnaît son génie."

"La naturaleza odia la uniformidad y ama la diversidad. Quizás ahí sea donde radica su genio."

"Le temps que vous lisiez ces lignes, sept cents millions de fourmis seront nées sur la planète. Sept cents millions d'individus dans une communauté estimée à un milliard de milliards, et qui a ses villes, sa hiérarchie, ses colonies, son langage, sa production industrielle, ses esclaves, ses mercenaires..."

"En el tiempo que necesita para leer estas líneas, 700 millones de hormigas habrán nacido en el planeta. 700 millones de individuos en una comunidad estimada a un billón de billones, que tiene sus ciudades, su jerarquía, sus colonias, su lenguaje, su producción industrial, sus esclavos, sus mercenarios..."

Bernard Werber, Les Fourmis

Bernard Werber, Las Hormigas

Abstract

Optimization problems have been intensively studied, especially since the development of operations research and computer science. Nowadays a huge number of complex problems are NP-hard, which implies that they cannot be optimally solved in a reasonable amount of time if the size of the problem is too large. This is the case for the Vehicle Routing Problem (VRP) that consists in finding a set of routes to serve a list of customers with a specific fleet of vehicles; a brute force approach is not efficient if the amount of customers is too large. Another example is the Course Timetabling Problem whose objective is to find a timetable with no clashes for students and teachers and that considers, most of the time, a lot of additional constraints such as teacher availability or room capacity.

Due to this complexity, several metaheuristics have been proposed to find near-optimal solutions that go from basic strategies, such as random search, to more sophisticated approaches, such as bio-inspired metaheuristics. An important component of the latter is the Genetic Algorithm (GA), which applies the selection processes found in nature to the solutions of an optimization problem. Another important bio-inspired metaheuristic is the Ant Colony Optimization (ACO), which is based on the foraging behavior of ants.

This dissertation is focused on how heuristic approaches and metaheuristics can be adapted to solve real optimization problems. The analysis and the adaptation of the ACO metaheuristic in particular is done for several combinatorial problems, among them Resource Allocation Problems and Scheduling Problems, such as Timetabling and VRP. Those problems are first modeled as Constraint Satisfaction Optimization Problems (CSOP). Then new heuristic algorithms, metaheuristics and hybrid metaheuristics are designed and applied to solve those CSOP. The results are then compared to different resolution methods such as GA or commercial tools in order to analyze their performance.

One of the problems studied in this dissertation consists in allocating students to classes in Swiss secondary schools, where students have different profiles due to their level in some fields or to the elective courses they attend. The pedagogical objective is to have a high diversity of profiles within a class and similarity between classes. In order to achieve this goal, the problem is modeled as a Resource Allocation Problem, where students are resources. The Resource Allocation Problem is then solved in two different ways, with a standard solver for CSOP, and with ACO. Eight real datasets are used to compare their performance. ACO provides better solutions than the

CSOP solver and in a shorter time. Results show that the pheromones used in ACO help to find better solutions in a much smaller amount of time.

The same problem has been modeled as a many-objective optimization problem, where the composition of classes combines pedagogical objectives with resource and economic objectives. In a many-objective problem, usually objectives are not compatible and improving one objective will penalize one or more of the other objectives. Therefore, there are several optimal solutions, whose set is called the Pareto front. To solve this many-objective problem, we adapt three approaches based on ACO, Harmony Search, and GA (NSGA-II). Eight real datasets from different schools are used and all approaches find several non-dominated solutions for each of them. Two methods are proposed to compare their performance, a simple one with a direct hypervolume comparison and a smoothing method where Pareto fronts are approximated with several runs for each approach. In four datasets, the three approaches are competitive, in the other four datasets NSGA-II outperforms the other approaches.

Resource allocation and timetabling problems can both be found in universities where students can select courses before or after the timetabling is done. In this dissertation, both problems, seat allocation and timetabling, are modeled separately and combined as CSOP. Two algorithms are designed and implemented to find a solution to both problems simultaneously maximizing the satisfaction of students using a CSOP solver and ACO for the timetabling problem. The results of both algorithms are then compared. Three real datasets from the Ecole Hôtelière de Lausanne have been used to carry out a complete experimental analysis. High quality solutions are obtained in a few minutes with both approaches; those solutions are currently used at the Ecole Hôtelière de Lausanne.

Another type of problem described in this dissertation is a complex VRP with many different constraints: time windows, heterogeneous fleet, multiple depots, multiple routes and incompatibilities. Five different approaches are presented and applied to fifteen real datasets. Those approaches are based on ACO and GA and are applied in their standard formulation and combined as hybrid metaheuristics, ACO-GA and GA-ACO. The results are compared regarding quality and computation time with two commercial tools. Considering the number of customers that cannot be served, one of the tools and ACO-GA outperform the others. Considering the cost, GA and GA-ACO provide better results. The computation time needed for one iteration is much better in GA and GA-ACO.

Resumen

Los problemas de optimización han sido intensamente estudiados, especialmente desde el desarrollo de la investigación operativa y la informática. Hoy en día una gran cantidad de problemas complejos son NP-hard, lo que implica que no pueden ser resueltos de forma óptima en un tiempo razonable si el tamaño del problema es demasiado grande. Este es el caso del Vehicle Routing Problem (VRP) que consiste en encontrar un conjunto de rutas para visitar una serie de clientes con una flota específica de vehículos; atacar el problema con fuerza bruta no es eficiente si el número de clientes es demasiado grande. Otro ejemplo es el Course Timetabling Problem cuyo objetivo es encontrar una planificación sin conflictos horarios para estudiantes o profesores, y teniendo en cuenta, muchas veces, restricciones adicionales como la disponibilidad de los profesores o la capacidad de las salas.

Debido a esta complejidad, se han desarrollado varias metaheurísticas para encontrar soluciones cercanas al óptimo, que van desde estrategias básicas, como la búsqueda aleatoria, hasta enfoques más sofisticados, como las metaheurísticas bio-inspiradas. Un ejemplo importante de estas metaheurísticas es Genetic Algorithm (GA) que aplica los procesos de selección que existen en la naturaleza a soluciones de un problema de optimización. Otra metaheurística bio-inspirada importante es Ant Colony Optimisation (ACO) basada en el comportamiento de las hormigas cuando buscan comida.

Esta tesis se centra en cómo adaptar las estrategias heurísticas y metaheurísticas para resolver problemas reales de optimización. Se analiza en particular ACO para varios problemas combinatorios, entre ellos Resource Allocation Problems (RAP) y Scheduling Problems, tales como la planificación de horarios y el VRP. En una primera etapa, se modelan esos problemas como Constraint Satisfaction Optimisation Problems (CSOP). En una segunda etapa se diseñan y aplican nuevos algoritmos heurísticos, metaheurísticas y metaheurísticas híbridas para resolver esos CSOP. Los resultados se comparan con otros métodos de resolución como GA o herramientas comerciales para analizar su desempeño.

Uno de los problemas analizados en esta tesis consiste en asignar alumnos a clases en colegios de Suiza, donde los alumnos tienen perfiles distintos debido a su nivel en algunas asignaturas o a las optativas a las que asisten. El objetivo pedagógico es tener una gran diversidad de perfiles dentro de una clase y similitud entre las distintas clases. Para lograr este objetivo, el problema se modela como un RAP, en el que los alumnos son recursos. El RAP se enfoca de dos maneras distintas,

con un solver estándar para CSOP y con ACO, y se compara el rendimiento de las dos estrategias mediante ocho conjuntos de datos reales. ACO proporciona mejores soluciones que el solver CSOP en un tiempo más corto. Los resultados muestran que las feromonas utilizadas en ACO ayudan a encontrar mejores soluciones en un tiempo mucho menor.

Ese mismo problema se modela como un problema de optimización multi-objetivo, incluyendo objetivos pedagógicos, económicos y de recursos. En un problema multi-objetivo, por lo general, los objetivos no son compatibles y la mejora de un objetivo penaliza uno o más de los otros objetivos. Por lo tanto, hay varias soluciones óptimas, cuyo conjunto se llama el frente de Pareto. Para resolver este problema multi-objetivo, se adaptan tres estrategias basadas en ACO, Harmony Search y GA (NSGA-II). Las tres estrategias encuentran varias soluciones no dominadas para cada uno de los ocho conjuntos de datos. Dos métodos permiten comparar su rendimiento, uno simple con una comparación directa de hipervolumen y un método suavizado donde los frentes de Pareto se aproximan con varias ejecuciones para cada estrategia. En cuatro conjuntos de datos, los tres métodos son competitivos, en los otros cuatro, NSGA-II supera los demás.

En las universidades en las que los estudiantes pueden seleccionar cursos antes o después de que se realice la planificación de horarios, se encuentran problemas de asignación de recursos y de planificación de horarios. En esta tesis, ambos problemas, asignación y planificación, se modelan por separado y combinados como CSOPs. Se diseñan e implementan dos algoritmos para encontrar una solución a ambos problemas simultáneamente maximizando la satisfacción de los estudiantes mediante un solver estándar de CSOP y un algoritmo ACO para el problema de planificación de horario. Para comparar los resultados de ambos algoritmos, se utilizan tres conjuntos de datos reales de la Ecole Hôtelière de Lausanne para llevar a cabo un análisis experimental completo. Soluciones de alta calidad se obtienen en pocos minutos con ambas estrategias; esas soluciones se utilizan actualmente en la Ecole Hôtelière de Lausanne.

Otro tipo de problema tratado en esta tesis es un VRP complejo con muchas restricciones diferentes: ventanas de tiempo, flota heterogénea, múltiples depósitos, múltiples rutas e incompatibilidades. Cinco estrategias diferentes se han aplicado a quince conjuntos de datos reales. Esas estrategias se basan en ACO y GA que se han aplicado en su formulación estándar y combinadas como metaheurísticas híbridas, ACO-GA y GA-ACO. Los resultados se han comparado en calidad y en tiempo de cálculo con dos herramientas comerciales. Considerando el número de clientes que no pueden ser servidos, una de las herramientas y ACO-GA superan a los demás. Teniendo en cuenta el coste, GA y GA-ACO proporcionan mejores resultados. El tiempo de computación necesario para una iteración es mucho mejor en GA y GA-ACO.

Agradecimientos

Los agradecimientos siempre se quedan para el final... y a estas horas de la noche, empieza a ser complicado juntar cuatro palabras para que tengan sentido, pero hay algunas personas muy queridas que tengo que mencionar aquí.

El primero, mi marido, Vincent, que nunca ha fallado. Niños, cenas, compras, y mucho más. Lo primero aguantarme durante estos años, y en especial este último año en el que le ha tocado pasar muchas horas, muchos días, e incluso varias semanas con los niños fuera de casa para dejarme trabajar tranquila en este proyecto.
Je t'aime mon coeur.

A mis niños, David y Mateo, que ya están más que hartos de los deberes de mamá que no puede ir al parque, ni de vacaciones, ni a veces leer el cuento para dormir. A partir de mañana, cariños, ya estaré mucho más con vosotros, para leer, jugar y para hablar también antes de dormir porque tenemos muchas *Parlations* de retraso.

A mes parents et à mes beaux-parents, qui ont toujours été là, prêts à s'occuper des enfants, de l'intendance de la maison et même de mon mari.

A David por haberme animado desde el principio, por haber encontrado siempre las palabras para darme más ganas de conseguirlo todavía, por haber contestado a mis correos nocturnos, de fines de semana y de vacaciones y por haber hecho realidad este proyecto a pesar de la distancia.

A Sonia por haber aguantado esos correos y esas llamadas fuera de horario laborable, por haber facilitado mis estancias en Madrid, por su cariño y por su amistad.

A mis compañeros de AIDA que siempre me han acogido en el laboratorio como una más.

A Javier por sus consejos sobre los problemas de optimización multi-objetivo.

A Jean-François Theubet pour avoir toujours eu le temps de m'expliquer avec patience les complexités du nouveau système éducatif du Canton de Vaud.

A mon équipe pour son soutien et sa compréhension dans ma dernière ligne droite, en particulier à Armine pour être bien plus qu'une collaboratrice.

A Fabien et à Pierre qui m'ont non seulement encougée, mais ont toujours cru en moi dans tous mes projets, y compris celui-ci.

A la Universidad Autónoma de Madrid por ofrecerme la oportunidad de realizar esta tesis doctoral.

A l'Ecole Hôtelière de Lausanne pour m'avoir permis de saisir l'opportunité de réaliser ce projet.

A todos vosotros, muchas gracias.

A vous tous, mille mercis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides a general overview of this PhD dissertation. Section 1.1 motivates the work carried out in this PhD dissertation. Section 1.2 describes the problem that has motivated this work and Section 1.3 contains the different objectives in which this dissertation is focused. The dissertation structure is then described in Section 1.4 and, finally, the main contributions and the associated publications are described in Section 1.5.

1.1 Motivation of the dissertation

A large number of complex problems can be found in the research literature. Those problems belong most of the time to fundamental (or classical) research or to industrial problems. An example of a classical research problem is the Travelling Salesman Problem (TSP) [4, 92, 99, 140]. The TSP consists of a list of cities, a salesman and a travel cost associated to each pair of cities. The objective of the problem is to find a sequence of those cities so that the salesman visits all cities and the total cost is minimized. The Graph Coloring Problem is also a classical problem that consists in minimizing the number of colors needed to color the vertices of a graph so that each vertex has a color that is different from the colors used for the vertices connected to it with an edge, [28]. Another example is the n-Queens problem that consists in placing n queens on a $n \times n$ chessboard so that none of them is threatened by the others.

In the industrial field, most of the problems have an optimization objective often related to quality or to cost, or to both of them. An example of such problems is the Vehicle Routing Problem (VRP), which is a generalization of the TSP. The VRP consists in finding a set of routes that serve a list of customers with a specific fleet of vehicles, [85, 87, 137, 147]. The Course Timetabling Problem (CTT) can also be classified as an industrial problem that is found in universities and schools, [31, 80, 104]. The objective of the standard CTT is to find a timetable with no clashes for the students that considers, most of the time, a lot of additional constraints that are specific to the school or to the university. The Resource Allocation Problem (RAP) is an additional example that consists in assigning a limited number of resources to objects that compete to get those resources. Objects can be activities as in communication systems [102, 154], or tasks as in scheduling problems [68, 95]. The resources are for example goods [67], people [68, 95], money [47], or seats in courses [17, 106]. Usually a RAP has a unique objective

such as maximizing satisfaction or productivity, but it can be multi-objective (MORAP) and have objectives that might be opposite. In asset management, for example, the objective is to maximize revenue, but the risk has to be minimized at the same time, [47]. A different example of MORAP can be found in staff scheduling, where the operational cost has to be minimized, but the working conditions have to be improved, [95].

Part of those problems are combinatorial optimization problems. A simple definition of such a problem states that a **combinatorial optimization problem** consists in finding in a finite set of solutions a solution or the solutions that optimize the objective of the problem. An important characteristic of those classical and industrial problems is that they might be **NP-hard problems**, which implies that the time needed to find a solution to those problems increases exponentially if the size of the problem increases linearly, [54, 152]. Therefore exact algorithms that provide systematically the best solution of the problem can be used only for small instances of the problem. For example, the best solution to the TSP can be found with an exact algorithm only if the number of cities is small. If there are n cities to visit, the number of possible solutions is $\frac{1}{2} \cdot (n-1)!$. With 6 cities, there are 60 solutions, with 10 cities there are 181'440 solutions, with 20 cities, there are 6×10^{16} solutions. An optimal solution to a problem with 85'600 cities was found in 2006, but this took 136 CPU-years, [4].

Due to this situation with NP-hard problems and to the importance of such problems in the research and in the industry, many heuristic algorithms and **metaheuristics** have been developed to find good solutions, if not the best, to those problems, [12, 15, 89]. The main difference with exact algorithms is that those approximation methods explore only part of the search space instead of the complete search space. As part of the solutions are not explored, the search is faster than with exact algorithms, but there is no guarantee that the best solution to the problem is found. The main advantage of those heuristic methods is that they can usually find a good solution in a relatively short time. Their main drawback is that this solution may be a local optimal solution, and the algorithm may have no opportunity to escape from the area of this solution in order to explore new areas in the search space.

The main difference between a heuristic algorithm and a metaheuristic is that the first one is often designed to solve a specific type of problem and is not adapted to other problem types. A metaheuristic on the other hand can be used to solve many types of problems, for example VRPs or CTTs. Ant Colony Optimization, Genetic Algorithms, Iterated Local Search, Simulated Annealing and Tabu Search are some examples of metaheuristics, [13].

Metaheuristics can be classified according to different characteristics, one of them being the number of solutions that are improved. Simulated Annealing and Iterated Local Search are examples of single solution metaheuristics, whose objective is to modify and improve the current solution. Ant Colony Optimization (ACO) and Genetic Algorithms (GA) are examples of **population-based metaheuristics**, where several solutions are used and contribute together to the search, [13].

Another category of metaheuristics is composed of the **bio-inspired metaheuristics**, whose design is inspired by systems found in nature. GA is an example based on the natural selection process whose idea was developed by Darwin. GA is a population-based metaheuristic whose individuals are going to evolve and improve through crossover, mutation and selection. ACO is also a bio-inspired metaheuristic based on the ants' foraging behavior. Harmony Search (HS), based on the way musicians compose harmonies, is an additional example that is sometimes classified as a bio-inspired metaheuristic, [55]. However the parallel between HS and music

composition is often criticized in the research community due partly to the fact that the human creativity process is difficult to represent with a simple metaheuristic.

Among the bio-inspired metaheuristics, the **swarm-based metaheuristics** are based on the intelligence of colonies or swarms of simple individuals that interact with each other and with their environment (e.g., ACO). The individuals apply very simple rules and through their interactions achieve a collective intelligent behavior, even if there is no centralized system that leads their behavior and their decisions. Apart from ACO, there are several swarm-based metaheuristics, such as Particle Swarm Optimization or Artificial Bee Colony, [14, 76].

Metaheuristics have also been hybridized to improve their performance, [12]. The **hybridization** consists in combining a metaheuristic with another optimization approach that might be another metaheuristic or come from a different field like exact algorithms.

1.2 Problem statement

The hybridization of metaheuristics and their application to real optimization problems is the main objective of this dissertation. Most of the benchmark datasets that can be found in the literature have characteristics that are much more complex in real-life problems. The benchmark datasets are very useful when the objective is to compare a novel approach to the existing ones. Nevertheless those approaches might be successful when applied to those benchmark datasets, but might not be adapted to real problems that can be found in the industry.

In this dissertation, different approaches, among them swarm-based and hybrid metaheuristics are applied to four types of problems. Three of them are real-life problems found in a business school, in Swiss secondary schools and in a logistics company. The fourth uses benchmark datasets for the university course timetabling problem.

Heuristic and ACO Approaches for Resource Allocation Problems

Chapter 3 presents and solves partly a real problem found in Swiss secondary schools.

The first problem analyzed in this dissertation is faced by secondary schools in the Canton de Vaud in Switzerland. It consists in allocating students to main classes. Students in the same main class attend part of the lessons together, but they are then split and grouped differently for some special subjects where the students' level is the grouping factor. As the pedagogical objective is to have main classes as mixed as possible regarding the different levels, the splitting-grouping process for the special subjects may lead to a scheduling issue. If the grouping of students coming from different main classes is not possible, the number of lessons and of teachers needed for those special subjects increases. It is therefore important to find an allocation of students to main classes so that the classes' composition is as mixed as possible, but it is also critical to ensure that this allocation enables then the splitting-grouping process minimizing the number of lessons and of teachers needed.

In the first part of this chapter, a complete constraint-based model is proposed for the problem that combines both the class allocation problem and the scheduling problem. In the second part, a model and two resolution approaches are presented for the allocation of students to classes in order to have classes as mixed and similar as possible. This allocation problem is

first modelled as a Constraint Satisfaction Optimization Problem (CSOP), [148]. The CSOP is then solved with a CSOP solver and with ACO, [38, 44, 136]. Eight datasets from different schools in Canton de Vaud are used to compare the performance of the two approaches.

This chapter also includes a special case of the Course Timetabling Problem that corresponds to the datasets of a benchmark developed for an international competition (ITC), [111]. The approach presented in this chapter is based on ACO completed with a local search based on a CSOP solver.

Heuristic and Hybrid Metaheuristic Approaches for RAP Combined with a Scheduling Problem

Chapter 4 presents and solves a real problem from a business school, the Ecole Hôtelière de Lausanne (EHL).

The second problem analyzed in this dissertation to test the efficiency of different approaches is a combinatorial optimization problem that exists in EHL: the allocation of elective courses to students depending on their preferences and on constraints related to the courses and their different combinations. There are several constraints that have to be satisfied: a student cannot be allocated to similar courses, courses have to be schedulable over a week, and some of them can be canceled if the demand is not sufficient. Therefore, the RAP is combined with a scheduling problem. The objective of the problem is to maximize students' satisfaction, which means to allocate them to their favorite courses.

In a first phase, random datasets with different sizes are generated for the simple RAP, which is solved with two approaches. The first one is a greedy algorithm that considers an order of students based on their grades and assigns students on a first-come first served basis, considering their preferences. The RAP is then modeled as a CSOP and the second approach uses an exact algorithm to solve the CSOP that runs for a limited amount of time.

In the second phase, the RAP is combined with the scheduling problem and two approaches are used to solve it. The combined problem is first modeled as a CSOP. The first approach is the same CSOP solver as in the simple RAP with a timeout. The second approach hybridizes ACO with the CSOP solver. Both methods are applied to three real datasets from EHL.

Comparison of Three Metaheuristics to Handle Many-objective RAP

Chapter 5 is based on the same problem as chapter 3, the class allocation problem existing in Swiss secondary schools.

The problem of the educational system in Canton de Vaud presented in chapter 3 is modeled as a many-objective resource allocation problem (MORAP). The seven objectives of this MORAP are related to the number of lessons, the number of teachers and the pedagogical goal. Three bio-inspired approaches are compared: ACO, HS and GA. The three of them are adapted to a many-objective problem, the GA adaption is NSGA-II, which is a well-known multi-objective metaheuristic. The three of them use the crowding distance value to sort the solutions, together with the successive Pareto fronts in NSGA-II and HS. This sorting is then used to guide the search: the best solutions deposit more pheromones in ACO, the worst solutions are removed from the population in NSGA-II and from the harmony memory in HS.

Hybrid Metaheuristics to Manage Complex Vehicle Routing Problems

Chapter 6 presents a real vehicle routing problem from a logistics company.

The complexity of this VRP lies in the fact that many constraints have to be considered to find a solution that can be used by the company, and in the fact that such a solution has to be found very quickly. A different set of several customers has to be served every day. The time is very short since the scheduling of those customers in routes and vehicles is done the day before at the very end of the day when most information is known and sent to the information systems of the company and to the vehicles' drivers. Due to the constraints, there is often a small number of solutions that include all customers. The company uses currently two commercial tools, one of them provides a solution very quickly, but is sometimes not able to include all customers in routes and some are not served. The second tool solves this problem, but the computation time varies a lot between days and the quality of the solution provided is not always good regarding costs.

Five different approaches are proposed to solve this VRP. Two of them are based on classical metaheuristics, ACO and GA. One is based on ACO and includes a local search. The other two approaches are hybrid metaheuristics that combine both of them, ACO and GA. In one of them, ACO-GA, ACO is the main metaheuristic and GA is used for local search. In the other, GA-ACO, GA is the main metaheuristic and ACO is used to break ties instead of using randomness when a decision has to be made in order to complete the solution. Those approaches and the two commercial tools are applied to fifteen real datasets in order to analyze the performance of the proposed hybrid metaheuristics.

1.3 Research Questions

As explained previously, there is a wide range of real complex problems that cannot be solved with exact algorithms and therefore need heuristic approaches to be solved. One of these heuristic approach is a swarm-based metaheuristic, Ant Colony Optimization. In order to be really efficient, the ACO algorithms have to be combined with a local search that intensifies the search in a promising area of the search space.

The main goal of this thesis can be formulated as the analysis and improvement of current models and metaheuristics based on ACO algorithms and their application to real combinatorial optimization problems.

This general research goal can be broken down into more specific research goals:

G1 To review the state of the art related to bio-inspired metaheuristics, mainly centered on Ant Colony Optimization, but including other approaches such as Genetic Algorithms and Harmony Search.

G2 To review the state of the art related to Constraint Satisfaction Problems and how they can be solved with metaheuristics.

G3 To review the state of the art in the different application fields analyzed in this dissertation: Resource Allocation Problems and Scheduling Problems, specially Timetabling and Vehicle Routing Problems.

G4 To develop different types of models for combinatorial optimization problems: Resource Allocation Problems and Scheduling Problems, specially Timetabling and Vehicle Routing Problems.

G5 To design new heuristic and metaheuristic approaches based on ACO.

G6 To hybridize metaheuristics and analyze their performance.

G7 To apply those new approaches and metaheuristics to real industrial problems.

G8 To compare their performance in those problems.

In order to reach those goals, three real optimization problems are used in this dissertation:

1. A course allocation problem in a business school;
2. A class allocation problem in secondary schools;
3. A vehicle routing problem in a logistics company.

Those three problems are modeled as Constraint Satisfaction Optimization Problems and different approaches are used to find solutions to them. In this dissertation, we also show how metaheuristics and hybrid metaheuristics can be applied to these real problems. The results obtained are compared with other traditional approaches and with commercial tools. Several real datasets from the three organizations have been used to perform this comparison.

1.4 Structure of the thesis

The dissertation has been structured in eight chapters. A brief description of each chapter's content is provided below.

- **Chapter 1: Introduction**. It provides a general background, context and motivations of this dissertation. The main objectives and research questions are stated, as well as the dissertation structure, main contributions and publications.
- **Chapter 2: State of the Art**. It introduces the State of the Art about bio-inspired metaheuristics and Constraint Satisfaction Problems. It also contains the State of the Art in three different industrial problems, RAP, MORAP and VRP.
- **Chapter 3: Heuristic and ACO Approaches for Resource Allocation Problems**. This chapter presents the allocation problem of students to classes in Swiss secondary schools. Two models are proposed, the first combines the class allocation problem with a scheduling problem. The second model includes only the allocation problem and is solved with two approaches, a CSOP solver and an ACO algorithm.
- **Chapter 4: Heuristic and Hybrid Metaheuristic Approaches for RAP Combined with a Scheduling Problem.** This chapter describes and solves a Course Allocation Problem combined with a Course Scheduling Problem. Two approaches are proposed

to solve the Course Allocation problem, a greedy algorithm and a CSOP approach. The CSOP approach is then adapted to solve the combined problem and compared to an ACO approach.

- **Chapter 5: Comparison of Three Metaheuristics to Handle Many-objective RAP**. In this chapter, ACO and HS are adapted to solve a many-objective RAP. Their performance is compared to the well-known NSGA-II. The data used to compare their performance come from Canton de Vaud and is the same as the data used in chapter 3.
- **Chapter 6: Hybrid Metaheuristics to Manage Complex Vehicle Routing Problems**. This chapter describes and solves a complex Vehicle Routing Problem with many different constraints: time windows, heterogeneous fleet, multiple depots, multiple routes and incompatibilities. Five different approaches are presented based on two metaheuristics, ACO and GA that are applied in their standard formulation and combined as hybrid metaheuristics. The results are compared with two commercial tools.
- **Chapter 7: Conclusions and Future Work**. The Research Questions described in Chapter 1 are addressed in order to provide some answers, based on the results obtained from this research.

1.5 Publications and Contributions

During the development of this work several publications have been generated. This section provides a short description of them, and the chapters where these contributions can be found in the thesis. Finally, these publications have been organized by journals and conferences, and sorted by year.

- Journals accepted
  - A.-M. Nogareda and D. Camacho. Optimizing satisfaction in a multi-courses allocation problem combined with a timetabling problem. *Soft Computing*, pages 1–10, 2016
    - Short summary: The resource allocation problem and the timetabling problem are traditional kinds of NP-hard problems. Both problems can be found in universities where students can select courses they would like to attend before or after the timetabling is done. When demand exceeds capacity, the universities may allocate the available seats independently from the timetabling, but students may have then to decide which courses they are going to attend because of clashes in their timetable. To avoid this situation, some universities prepare their timetable considering students' selection. In addition to that, students may submit preferences over courses, and the school administration has to assign seats and do the timetable considering both preferences and clashes. In this paper, both problems, seat allocation and timetabling, have been modeled separately and combined as Constraint Satisfaction Optimization Problems (CSOP). Two algorithms have been designed and implemented to find a solution to both problems simultaneously maximizing the satisfaction of students using a CSOP solver and an Ant Colony Optimization algorithm for the timetabling problem. The results of both

algorithms are then compared. The allocation and timetabling procedures are based on preferences for courses defined by students, and on the administration's constraints at the Ecole Hôtelière de Lausanne. Three real data sets have been used to carry out a complete experimental analysis. High quality solutions are obtained in a few minutes with both approaches; those solutions are currently used at the Ecole Hôtelière de Lausanne.

* JCR: Q2, 2015 Impact factor: 1.630

* The contribution of this paper is fundamental for Chapter 4. It includes two approaches used to solve the problem that combines the Course Allocation and the Course Timetabling.

– A.-M. Nogareda and D. Camacho. A constraint-based approach for classes setting-up problems in secondary schools. *International Journal of Simulation Modelling (IJSIMM)*, 16(2), 2017

* Short summary: In this paper, we present the problem of allocating students to classes in Swiss secondary schools, where students have different profiles due to their level in some fields or to the options they attend. The pedagogical objective is to have a high diversity of profiles within a class and similarity between classes. In order to achieve this goal, the problem is modelled as a resource allocation problem (RAP), where students are resources, using a constraint satisfaction optimization approach (CSOP). The RAP is then solved in two different ways, with a solver for CSOP, and with an ant colony optimization algorithm (ACO). Eight real datasets are used to compare their performance. The ACO algorithm provides better solutions than the CSOP solver in a shorter time. Results show that the pheromones used in the ACO help to find better solutions in a much smaller amount of time. The short computation time enables the school's directors to simulate different compositions of their future classes before having the final results of the last exams.

* JCR: Q2, 2015 Impact factor: 1.683

* The contribution of this paper is fundamental for Chapter 5. It includes the approaches based on ACO and on CSOP used to solve the problem of class allocation.

- Journals under review

– A.-M. Nogareda, D. Camacho, and J. Del Ser. A comparison of bio-inspired heuristics applied to many-objective resource allocation problems. *Soft Computing*, Submitted 2017

* Short summary: In this article, we present a many-objective optimization problem found in secondary education in Switzerland, where the composition of

classes combines pedagogical objectives with resource and economic objectives. In this type of problem, it is usually not possible to find one optimal solution since, most of the time, objectives are not compatible and improving one objective will penalize one or more of the other objectives. Therefore, there are several optimal solutions to the problem, whose set is called the Pareto front. To solve this many-objective problem, we adapt three multi-objective approaches, based on the ant colony optimization algorithm (ACO), on the harmony search algorithm (HS), and on an evolutionary algorithm (NSGA-II). Eight real datasets from different schools were used and all our approaches find several non-dominated solutions for each of them. Two methods are used to compare the performance of those approaches, a simple one with a direct hypervolume comparison and a smoothing method where Pareto fronts are approximated with several runs for each approach. In four datasets, the three approaches are competitive, in the other four datasets NSGA-II outperforms the other approaches.

  * JCR: Q2, 2015 Impact factor: 1.630

  * The contribution of this paper is fundamental for Chapter 5. It introduces the class allocation problem as a MORAP and the three different approaches to solve it based on ACO, GA and HS. It also includes the hypervolume indicator and a variant of it, the Smoothing Hypervolume, to compare the results of the three approaches.

- A.-M. Nogareda, J. Del Ser, and D. Camacho. Hybrid metaheuristics to manage complex vehicle routing problems. *Engineering Applications of Artificial Intelligence*, Submitted 2017

  * Short summary: This paper addresses a multi-attribute Vehicle Routing Problem, the maVRP, with time constraints, heterogeneous fleet, multiple depots, multiple routes and incompatibilities of goods. Four different approaches are presented and applied to fifteen real datasets. They are based on two metaheuristics, Ant Colony Optimization (ACO) and Genetic Algorithm (GA), that are applied in their standard formulation and combined as hybrid metaheuristics to solve the problem. As such ACO-GA is a hybrid metaheuristic using ACO as main approach and GA as local search. GA-ACO is a memetic algorithm using GA as main approach and ACO as local search. The results regarding quality and computation time are compared with two commercial tools currently used to solve the problem. Considering the number of customers served, one of the tools and the ACO-GA approach outperform the others. Considering the cost, ACO, GA and GA-ACO provide better results. Regarding computation time, GA and GA-ACO are the most competitive among the benchmark.

  * JCR: Q1, 2015 Impact factor: 2.368

  * The contribution of this paper is fundamental for Chapter 6. It presents the VRP and four different approaches based on ACO and GA. It also includes the results of two commercial tools in order to compare the performance of the proposed approaches.

- Conferences
  - A. M. Nogareda and D. Camacho. Integration of ant colony optimization algorithms with gecode. In *Principles and Practice of Constraint Programming (Doctoral Program CP 2014), International Conference on*, pages 59–64, 2014

    * Short summary: The objective of this study is to integrate an Ant Colony Optimization (ACO) algorithm with a Constraint Satisfaction Problem (CSP) model using Gecode, which is a C++ library developed to solve CSPs. The resulting hybrid heuristic will be tested with several problems of Course Timetabling: some benchmark examples and a real problem. The combination of ACO and CSP should produce interesting results since ACO algorithms are efficient when dealing with huge problems and CSPs models are well-adapted to handle hard constraints. Another goal is to test different variants of this heuristic in order to tune its components and analyze their performance.

    * Core: A

    * The contribution of this paper is fundamental for Section 3.10. It introduces the Course Timetabling Problem and the ACO approach applied to it.

  - A. M. Nogareda and D. Camacho. Constraint-based model design for timetabling problems in secondary schools. In *Innovations in Intelligent SysTems and Applications (INISTA), 2015 International Symposium on*, pages 1–6. IEEE, 2015

    * Short summary: In this paper, we propose a constraint-based model for the combination of a course timetabling problem and a course allocation problem for secondary schools in Switzerland. Course timetabling has been widely studied for Universities, but for schools, solutions have only been proposed for specific countries or even for specific schools. In fact, timetabling problems present the difficulty of being case-specific through specific satisfaction constraints. In addition, due to a recent reform of the education system in Switzerland, the timetabling problem is combined with a course allocation problem that has an important impact on timetables. Indeed, some topics are defined by a curriculum but each student may be assigned to up to five different options depending on past grades and student choices. Both problems, timetabling and allocation must be solved simultaneously and educational objectives are related to both composition of classes and timetables of students. A description of the problem and the educational objectives to consider are presented. Finally, a complete constraint-based model based on hard and soft constraints has been designed to solve the combined problem, both types of constraints are described in detail and a qualitative complexity analysis of this model is given.

    * Core: C

  * The contribution of this paper is fundamental for Chapter 3. It introduces the complete model for the class allocation problem combined with a timetabling problem.

- A.-M. Nogareda and D. Camacho. Optimizing satisfaction in a multi-courses allocation problem. In *Intelligent Distributed Computing IX*, pages 247–256. Springer, 2016

  * Short summary: The resource allocation problem is a traditional kind of NP-hard problem. One of its application domains is the allocation of educational resources. In most universities and business schools today, students select the courses they would like to attend by ranking the proposed courses. However, to ensure the quality of a course, the number of seats is limited, so not all students can enroll in their preferred courses. Therefore, the school administration needs some mechanism to assign the available resources as soon as possible, trying to optimize the students' wishes. In this paper, the course allocation problem has been modeled as a Constraint Satisfaction Optimization Problem (CSOP) and two metrics have been defined to quantify the satisfaction of students. The problem is solved with Gecode, and its results are compared with a greedy-based algorithm showing how the CSP approach is able to optimize the allocation of resources optimizing the students' satisfaction. Another contribution of this work is related to the possibility to allocate simultaneously several courses, generating feasible solutions in a short time. The allocation procedures are based on preferences for courses defined by students, and on the administration's constraints that define the available resources at the Ecole Hôtelière de Lausanne. Ten datasets have been generated using the distribution of preferences of students for courses, and a complete experimental analysis has been carried out using these datasets evaluating the performance of the algorithms considered.

  * The contribution of this paper is fundamental for Chapter 4. It includes the two approaches used to solve the course allocation problem, the greedy algorithm and the CSOP solver.

Chapter 2

State of the Art

This chapter is dedicated to the current state of the art in the main research fields that compose the heart of this thesis. The first part of it expounds three bio-inspired algorithms; it starts with an overview of a swarm-based algorithm, Ant Colony Optimization, which is a recurrent theme in this thesis, and is completed with two other population-based algorithms, Genetic Algorithm and Harmony Search.

The chapter continues with the definition of Constraint Satisfaction Problems and with several examples of application of this theory. This part is completed with two approaches used to solve such problems.

It continues with the Resource Allocation Problem that includes two special cases of this problem. The first case is about the allocation of courses to students in universities, and the second one deals with the course timetabling problem, where several resource types have to be allocated to courses: timeslots and rooms. This topic is then generalized to the Many-objective Resource Allocation Problem that includes a metric used to compare different resolution approaches.

The last section of this chapter describes the Vehicle Routing Problem, different variants of it and the different approaches that have been applied to this classical problem.

2.1 Bio-inspired Metaheuristics

The research community has been trying and addressing optimization problems for several centuries. In the 17th century, Pierre de Fermat worked on the minimization and maximization of mathematical functions. Since then, this research field has been intensively studied, especially in the 20th century with the development of operations research during the World Wars and, some years later, with the development of computer science. The first optimization methods were mostly based on mathematical theories and on exact methods [152]. Nevertheless, as the complexity of the studied problems increased, those exact methods became difficult to use because of the time needed to find optimal solutions. Stochastic or heuristic search became popular in the middle of the 20th century.

The complexity of the problems increased with the industrial evolution of our society. Nowa-

days a huge number of the complex problems found in the research community and in the industry are NP-hard, which implies that they cannot be optimally solved in a reasonable amount of time if the size of the problem is too large. This is the case, for example, for the Vehicle Routing Problem that consists in finding a set of routes that serve a list of customers with a specific fleet of vehicles. If the number of customers is small, an optimal solution can easily be found, but if the amount of customers is too large, a brute force approach is not efficient [87]. Another example is the Course Timetabling Problem whose objective is to find a timetable with no clashes for students and teachers and that considers, most of the time, a lot of additional constraints such as teacher availability or room capacity [89].

More formally, NP-hard stands for Non-deterministic Polynomial time. The validation of a solution of an NP-hard problem can be done very quickly, in a polynomial time. But there is currently no efficient method that would find an optimal solution to the problem in a reasonable amount of time [54, 87].

Due to this complexity, several heuristic approaches and metaheuristics have been proposed to find near-optimal solutions to those problems that range from basic strategies, such as random search, to more sophisticated approaches, such as **bio-inspired metaheuristics** [13, 15, 38]. The latter method has its origins in Evolutionary Computing that are inspired by the natural evolution theory of Darwin. An important component of this area is Genetic Algorithm (GA) that became popular in the 1970s [130, 151]. Those approaches apply the selection processes found in nature to the solutions of a problem by combining and modifying them through crossover, mutation and selection operators. Another important bio-inspired metaheuristic is Ant Colony Optimization (ACO) based on the foraging behavior of ants [45, 44, 43, 94]. When they are looking for food, ants deposit pheromones on the paths they use to inform other ants that they have used this path. This pheromone concept is used in ACO as probabilities that guide the search to and through areas of the search space that are interesting and promising because good solutions have been previously found there.

Other examples of bio-inspired metaheuristics are Particle Swarm Optimization (PSO), Coral Reef Optimization (CRO) or Harmony Search (HS). PSO is based on the behavior of bird flocks or fish schools and defines a position and a velocity for each particle; the position corresponds to a possible solution and the velocity contains information about how this solution is going to evolve [25, 49, 144]. CRO is based on coral reefs' reproduction; after reproduction, corals, that correspond to possible solutions, must fight to find a space in the reef, some of them die because of this fight and this corresponds to the evolution of the population [126]. HS is inspired from the composition process of musicians; harmonies correspond to possible solutions and new solutions are developed using existing harmonies or randomness [55, 84, 96].

The difference between a heuristic algorithm and a metaheuristic lies in the fact that a heuristic algorithm is often designed to solve a specific problem, but is usually not adapted to any other type of problem. On the contrary, a metaheuristic is a stochastic approach independent from any type of problem that defines a framework with a set of rules. The rules in GA describe the operations that can be performed on existing solutions in order to find new solutions to the problem. The rules in ACO explain how the pheromones can be used to calculate the probability to select a component of the solution. A metaheuristic is thus a strategy that enables a guided search through the complete space search with the objective of finding (near) optimal solutions, but with no guarantee that the optimal solution will be found [12, 15].

The metaheuristics can be classified into different categories [13, 15]. Some of them are

population-based metaheuristics, as the bio-inspired examples mentioned above. They are based on a population of solutions: particles in PSO, ants in ACO, chromosomes in GA, corals in CRO and harmonies in HS. Some of them rely on the evolution of those populations, the individuals interact with each other to generate new individuals, as GA, CRO or HS. Some of them make use of individuals that look for a solution on their own independently from the others, those metaheuristics are the swarm-based metaheuristics, as ACO and PSO, the intelligence depends on the collective behavior of individuals.

Another category is the local search metaheuristics which use a **single solution** that will be changed iteratively with usually small moves. This category uses the neighborhood of a solution that contains all the solutions that can be reached through these small moves. Simulated annealing is an example that uses a move strategy similar to the annealing process in metallurgy that consists in controlling the heating and cooling of a material in order to reduce its defects [81]. Tabu search uses a memory to store past moves that are forbidden for a given number of steps during the search [58].

Hybrid metaheuristics have also been proposed to improve their performance [12, 40, 52, 59, 121, 150]. The hybridization consists in combining a metaheuristic with another optimization approach that might be another metaheuristic or come from a different field like exact algorithms. ACO often uses a local search component in order to improve the quality of the solutions found by the ants, the same occurs to GA. Both metaheuristics are efficient when they explore the search space looking for a promising area in it, but lack sometimes efficiency when they want to explore in a more systematic way a promising area. When GA is combined with such a local search, the resulting hybrid metaheuristic is often called memetic algorithm [118].

Those metaheuristics are applied to many different types of problems, among them NP-hard problems [5, 66, 89]. Examples can be found in classical problems, as the n-Queens problem. But they are also used to solve real complex problems, as the Resource Allocation Problem whose objective is to assign a limited number of resources to objects that compete to get those resources [49, 113]. Part of those problems are combinatorial optimization problems whose objective is to find in a finite set of solutions a solution or the solutions that optimize the objective of the problem.

In the rest of the current section, three bio-inspired metaheuristics are presented with more details, Ant Colony Optimization, Genetic Algorithm and Harmony Search.

2.1.1 Ant Colony Optimization (ACO)

The **swarm-based metaheuristics** are part of the population-based metaheuristics. They are inspired from the behavior of entities that have simple rules as individuals, but a very complex interaction level between individuals. Among those swarm-based metaheuristics, the most popular are ACO inspired by the ants' foraging behavior and PSO inspired by birds' behavior.

This section is dedicated to ACO. Section 2.1.1.1 presents the first ACO algorithm and two variants of it are presented in Sections 2.1.1.2 and 2.1.1.3. Section 2.1.1.5 gives a few examples of applications of ACO algorithms.

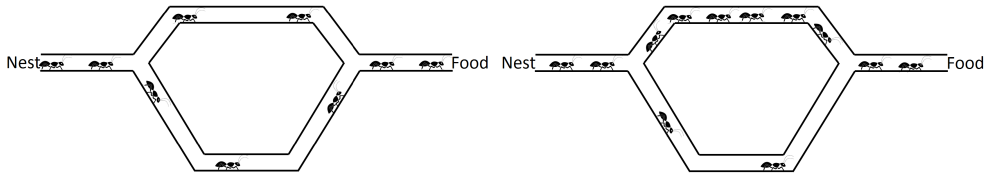

Figure 2.1: Ants behavior. Left: At the beginning, no pheromone is present on the paths, the selection is random. Right: After some time, more pheromone is present on the shortest path, more ants select this path.

2.1.1.1 Ant System

As explained in [44, 136], **Ant System** (AS), the first ACO algorithm introduced by Dorigo in 1992, is a probabilistic algorithm based on the foraging behavior of real ants. When an ant has found food, it deposits a **pheromone** on its way back to the nest. This pheromone is then detected by other ants that prefer to follow trails where more pheromone was deposited. However, pheromones evaporate over time, which implies that either more ants deposit pheromone on a trail or the pheromone on this trail disappears. Ants follow thus the shortest path to reach the food source since on this path ants go faster and therefore more pheromone is deposited than on the longer paths.

This behavior was shown with a real experimentation in 1989, where a colony of ants that had to go from their nest to the food source [65]. The ants had to select between two paths, one of them being shorter. At the beginning of the experimentation, the selection of the path was completely random, but after some time, more and more ants selected the shorter path. Figure 2.1 illustrates this behavior.

This problem is similar to finding the shortest path between two vertices in a graph. In an ACO algorithm, a specific number of artificial ants build a solution step by step selecting the next edge considering the quantity of pheromone deposited there by the previous ants. If more pheromone is deposited, the probability is higher that the ant selects this edge. When this ant finds a solution, it deposits pheromone on the edges it has selected before, whose quantity may depend on the quality of the solution it found.

In addition to the pheromones, artificial ants may use heuristic information, the **visibility**, that depends on the analyzed problem, to select a path. For instance, in the Traveling Salesman Problem, the visibility may be related to the distance from the current city to the other unvisited cities. The probability that in step t the city i is connected to the city j, which means that the arc (i, j) is selected, by ant k is thus given in AS by Expression 2.1 [45].

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t) \cdot \eta_{ij}^\beta}{\sum_{u \in \mathcal{N}_i^k} \tau_{iu}^\alpha(t) \cdot \eta_{iu}^\beta} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{if } j \notin \mathcal{N}_i^k \end{cases} \tag{2.1}$$

Where $\tau_{ij}(t)$ is the quantity of pheromones on arc (i, j) in step t, η_{ij} is the visibility of arc (i, j), $\mathcal{N}_i^k$ is the list of cities that can be connected to the city i, and α and β are two parameters that balance the importance of the pheromones' quantity versus the visibility. If the pheromones

become more important, then a promising area of found solutions is deeper explored. On the other hand the visibility enables to explore new areas that might be promising as well.

At the end of a complete iteration, that is when all ants have built a solution, the quantity of pheromones on each path, that is on each arc (i, j), has to be updated as in Expression 2.2

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \tag{2.2}$$

where

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{\varphi}{L_k} & \text{if ant } k \text{ has selected arc } (i,j) \text{ in step t} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Where m is the number of ants, ρ is the evaporation rate, whose objective is to avoid the convergence to a local optimal solution, φ is a constant and L_k is an indicator of the quality of the solution, for example the length of the tour in the case of the TSP.

An ACO algorithm can be seen as a construction metaheuristic, which is a method that is going to build a solution step-by-step. At each step, each ant builds a new part of the solution by selecting an arc, but does not try to improve or change what has already been built in the previous steps.

Algorithm 1 represents the simplest ACO algorithm where at each iteration, solutions are built by each ant of the colony and pheromones are updated before the next iteration starts.

Algorithm 1: The basic ACO

```
1 Initialize pheromones, visibility and probabilities
2 for i = 1, ..., NbIterations do
3 |   Solutions are built by ants
4 |   Local search is applied if needed
5 |   Pheromones are updated
```

2.1.1.2 Elitist strategy

The **elitist strategy** consists in giving a higher importance to the global best solution found in the previous iterations [45]. If the value of the indicator of the quality of the best solution found so far is $L*$, then pheromones are updated at the end of an iteration as in Expression 2.4

$$\tau_{ij} = \rho \cdot \tau_{ij} + \Delta\tau_{ij} + e \cdot \frac{\varphi}{L*} \tag{2.4}$$

Where e is the number of elitist ants. If more elitists ants are considered, more importance is given to the best solution found so far. But if e is too large, then the global best solution leads all ants to find a solution in the neighborhood of it and after some time, all ants find exactly the same solution.

2.1.1.3 Max-Min Ant System

The difference between AS and the **Max-Min Ant System** (MMAS) depends on the pheromone update. While in AS, all ants update the pheromones at each iteration, in MMAS, this is no longer the case [141].

In MMAS, only the ant that found the best solution of the iteration deposits pheromone on the trails, this best solution being the one from the current iteration or the global one, as in Expression 2.5. This characteristic allows to exploit more the best solutions by exploiting more their neighborhood and searching a more promising area of the search space.

$$\tau_{ij} = \rho \cdot \tau_{ij} + \Delta\tau_{ij}^{best} \tag{2.5}$$

Another characteristic of MMAS is the fact that the quantity of pheromones deposited on a trail is bounded, that is limited to a maximum τ_{max} and a minimum τ_{min}. If, after the update, τ_{ij} is greater than τ_{max}, its value is reduced and $\tau_{ij} = \tau_{max}$. If, after the update, τ_{ij} is smaller than τ_{min}, its value is increased and $\tau_{ij} = \tau_{min}$. These limits for the pheromones avoid the stagnation of the search in the neighborhood of the best solution and enable the exploration of other areas in the search space.

At the beginning of the search, the pheromones are initialized to the maximum τ_{max}, this initialization is performed again from time to time during the iterations in order to avoid local optimal solutions.

2.1.1.4 Local search

In combinatorial problems, such as TSP, ACO algorithms are more efficient when combined with a **local search** that improves the solutions found by the ants [94, 99, 140].

Local search approaches are neighborhood searches where the initial solution is improved by small changes [93, 140]. These small changes can be performed until the best solution in the neighborhood is found or only once.

In [140], to solve TSP, the local search mechanism is the 2-opt heuristic that consists in exchanging 2 edges, and the 3-opt mechanism that consists in exchanging 3 edges. This local search is applied to all solutions found by the ants or only to the best solution of the iteration.

Tabu search is another popular local search combined with ACO algorithms [93, 155]. In a tabu search approach, a small change can be accepted even if the solution becomes worse, for example to escape from a local optimum. In order to avoid to go back to this local optimum, a tabu list of solutions is kept and changes that would lead to a tabu solution are forbidden.

2.1.1.5 Applications of ACO algorithms

ACO algorithms have often been used to solve different types of optimization problems. In the early years of this metaheuristic, most of the problems where ACO was used could be represented

with a graph, such as the Traveling Salesman Problem [140] or the Vehicle Routing Problem [10, 153].

Very quickly ACO algorithms spread across the research community and were used in many different fields, for example ACO has been used to define the location of hubs in a network and to allocate each node of this network to one of the hubs [122]. ACO algorithms have also been adapted to solve a multi-objective resource allocation problem [23], a timetabling problem [133] or a course allocation problem [106].

ACO algorithms have also been applied to classical problems, such as the Graph Coloring Problem [28, 46] or the Quadratic Assignment Problem [53, 142].

Another field of application of ACO algorithms is video games [63, 64]. The objectives in this type of problem can be very different, from automatically solving a level of the game to generating automatically a new level.

2.1.2 Genetic Algorithms (GA)

Considering Darwin's evolution theory, a population evolves considering four main characteristics among others:

- Genes are used to code and store the characteristics of individuals. The genotype of an individual will thus contain all those characteristics.
- The individuals fit more or less in their environment depending on their genotype.
- One or several individuals together will generate new individuals that inherit characteristics of the generators through the inheritance of genes.
- The individuals that fit more in their environment reproduce themselves more than the ones that are not adapted. The new individuals should thus be more adapted to the environment than the previous generations.

Those characteristics are the basic fundamentals of GA that were introduced by John Holland [71] and that can be used, as ACO, to solve optimization problems [1, 7, 151].

GA start with an initial population of individuals that is usually generated randomly. Each individual corresponds to one solution of the optimization problem that has to be solved. This solution has thus to be representable with a genotype which is usually coded as a linear vector of numerical values. For example, in the case of a TSP, this vector can be the sequence of the cities that are visited in the corresponding solution, each gene is thus a position in the sequence.

In order to evaluate a solution, that is the fitness of the individual in the environment, GA need an evaluation function, often called **fitness function**. The objective of the optimization problem is to minimize or to maximize this fitness function. Different solutions can thus be compared to each other, the ones with a better fitness function fit better in their environment.

The existing solutions in the population, often referred to as parent population, are used to create new individuals, often called children or offspring solutions. This generation of solutions is based on the following steps:

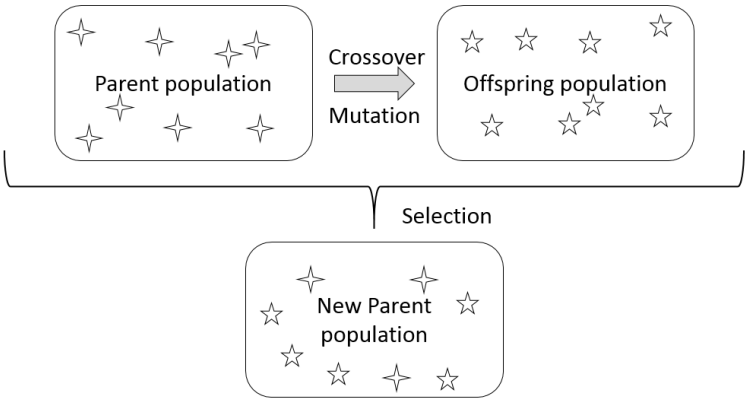

Figure 2.2: Genetic algorithm. The initial parent population generates an offspring population and then a new parent population is selected among them.

- **Selection of parents**: One or more solutions are selected among the parent population. The selection process is described in Section 2.1.2.1.

- **Reproduction**: The chosen parent solutions generate a new individual through reproduction that consists in two main operators:

  - **Crossover**: When two or more solutions are selected among the parent population, they are combined together in order to generate a new individual.
  - **Mutation**: A single solution may be selected, it generates a new individual through a mutation process.

  The reproduction process is described in Section 2.1.2.2.

- **Replacement**: Individuals in the parent population may be replaced by new individuals generated by reproduction. The replacement process is described in Section 2.1.2.3.

In all steps, two important factors have to be considered. The **diversity** of the population is critical to ensure that the search space is explored in an effective way. If the diversity is reduced too early in the search process, this one might be stuck in a search area that does not contain the optimal solution. On the other hand, the **selection pressure** gives a higher chance to be selected to the individuals that have a better fitness function. Increasing the selection pressure leads to a high convergence rate, but diversity decreases and the solution found might be a local optimal. Reducing the selection pressure favors diversity, but the search may then become random if the fitness function is not considered.

Figure 2.2 represents the steps of the genetic algorithm: the initial parent population generates an offspring population and then a new parent population is selected among both populations. Algorithm 2 represents the steps of a simple genetic algorithm. The initial population is generated randomly, then at each step of the process, the fitness function is evaluated for all individuals and a new population is created with offspring solutions.

Algorithm 2: Simple Genetic Algorithm

```
Initialize population P randomly
for i = 1, ..., NbIterations do
    Evaluate fitness function for all individuals I ∈ P
    P_new ← ∅
    while P_new is not complete do
        Select parents in P
        Generate an offspring solution I_new with the selected parents
        P_new ← P_new ∪ {I_new}
    Select the solutions for P in P ∪ P_new
```

2.1.2.1 Selection of parents

The selection of parents might be based on several methods. Two of them, proportionality and tournament, are explained in this section.

The **proportionality** rule calculates the probability $p(I_i)$ of each solution I_i to be selected using the fitness value of I_i. A solution with a better fitness should have more probability to be selected than a worse solution, for example as in Expression 2.6, where n is the size of the population. The best solution with the highest fitness value has the highest probability to be selected.

$$p(I_i) = \begin{cases} \dfrac{f(I_i)}{\sum_{l=1}^{n} f(I_l)} & \text{If the objective is to maximize the fitness function} \\ 1 - \dfrac{f(I_i)}{\sum_{l=1}^{n} f(I_l)} & \text{If the objective is to minimize the fitness function} \end{cases} \tag{2.6}$$

In the **tournament** method, a set of n_t solutions are randomly selected among the n solutions of the population, $n_t \leq n$. This selection can be with or without replacement, which means that one solution can be selected several times or only once respectively. The chosen parent will then be the best solution of this set of n_t solutions.

2.1.2.2 Reproduction

The reproduction process generates the new individual, the offspring solution, with the selected parents. Two operators are used in GA, the crossover and the mutation. Both of them are explained in this section.

The **crossover** consists in using the genes of two or more individuals to generate the offspring solution. Several methods that use two parents are described here.

The simplest crossover is the one-point crossover [71]. As the solution corresponding to one individual is represented with a linear vector of numerical values, this technique will randomly select a crossover point and exchange the beginning and the end genes of both parents as shown

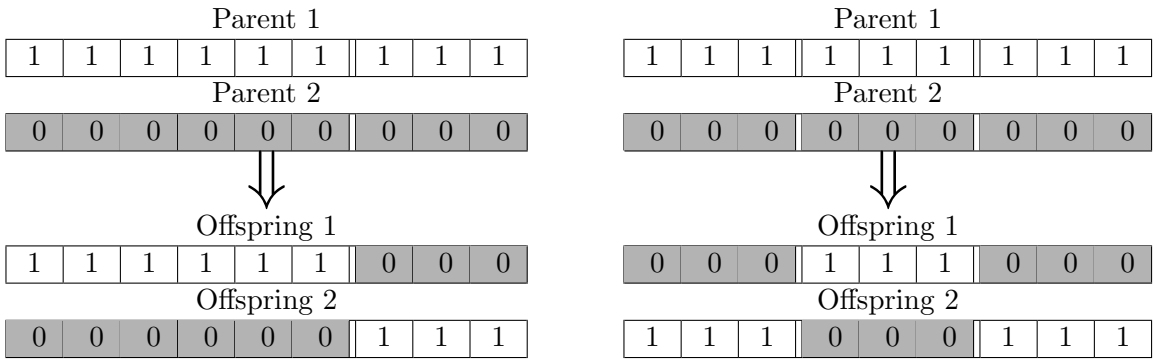

Figure 2.3: Generation of offsprings through k-point crossover. Left: One-point crossover. Right: Two-points crossover.

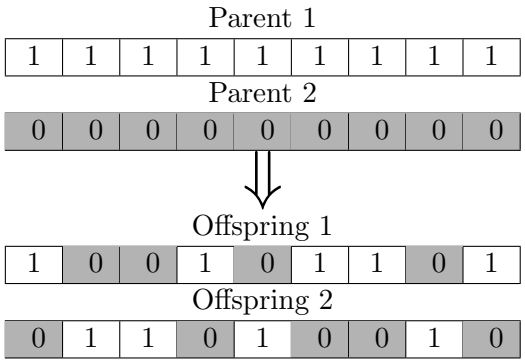

Figure 2.4: Generation of offsprings through uniform crossover.

in the left illustration of figure 2.3. The left part of parent 1 is completed with the right part of parent 2 in order to create offspring 1, and offspring 2 is generated with the remaining genes of both parents.

The two-points crossover is similar to the one-point crossover, but two crossover points are used as in the right illustration of figure 2.3. The parent that supplies the genes changes at each crossover point.

Another crossover technique, called the uniform crossover, consists in using each gene individually instead of using sequences of genes. Each gene of the offspring will come from one of the parents with a probability that is usually 0.5. Each gene of a parent has thus a probability of 0.5 to become part of the offspring.

The **mutation** operator is used to randomly change the value of genes with a mutation probability. It introduces in this way new values for genes that may not exist in the population. As crossover uses existing genes in the population, this might lead to a situation where diversity is lost and genes have the same value in all solutions of the population. The mutation operator ensures that the exploration of the complete search space is possible. Nevertheless if the mutation probability is too high, the search might not converge to the optimal solution [61]. With a low mutation probability, the selection pressure is high, but the search might converge to a local optimum. Figure 2.5 illustrates the mutation of a solution. One gene is randomly selected and

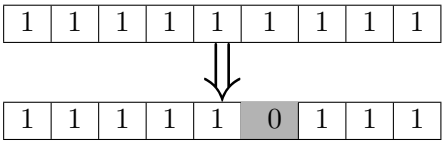

Figure 2.5: Mutation.

its value is changed.

2.1.2.3 Replacement

Once the offspring population is generated, a method has to be implemented in order to decide which individuals will be kept in the population among the parent and the offspring solutions [130, 149]. In the **generational GA**, the offspring population completely replaces the parent population as in algorithm 2. In the **steady-state GA**, each generated offspring solution might replace an individual in the parent population. A strategy for the selection of the replaced individual has thus to be defined, it can be for example the parent or the worst solution in the population or the oldest one. Replacing the worst solution increases the selection pressure.

2.1.3 Harmony Search (HS)

Harmony Search (HS) is a metaheuristic approach proposed in 2001 to solve optimization problems [56, 55, 96]. It is inspired from the music improvisation process whose objective is to find a pleasant harmony. A harmony is composed when each musician plays a note with their instrument. For example, in a trio of musicians, each musician plays a note and the trio of those notes composes a harmony.

HS is a population-based metaheuristic, algorithm 3 represents the steps of the HS. The population is stored in the **harmony memory** (HM) that contains harmonies already created. This population has to be initialized in the same way as the initial population in GA. Then step-by-step, new harmonies are improvised using HM, each musician selects an existing note in HM or generates it randomly from all possible notes of their instrument. If this new harmony is better than the worst harmony in HM, it replaces it. If not, it is rejected.

The **Harmony Memory Considering Rate** (HMCR) is the parameter that determines if the note selected by the musician comes from HM or is randomly generated. The HMCR ranges from 0 to 1 and a value of 0.95 means that the new note has a probability of 95% to come from the HM and of 5% to be randomly generated.

The second parameter of HS is the **Pitch Adjusting Rate** (PAR) which allows to tune the note to a note that is close to the one initially generated. A value of 0.1 for the PAR means that there is a probability of 10% that the generated note is changed to a neighboring note.

In HS, each note corresponds to a component of the solution and a harmony is a solution to the problem. As in GA, a solution can be a linear vector of numerical values. When a new solution is created, for each variable of this vector, there is a probability of HMCR to select

its value among the values already assigned to it in an existing harmony in HM. There is a probability of 1-HMCR to assign a random value to this variable. If the value comes from HM, then this value is changed to the closest upper value with a probability PAR/2 or to the closest lower value with a probability PAR/2.

Algorithm 3: Harmony Search Algorithm

```
1  Populate HM with random allocations
2  for i = 1, ..., N do
3      for Each variable do
4          if rand(0,1) < HMCR then
5              Select randomly a solution in HM
6              Assign to the variable its value in this solution
7              if rand(0,1) < PAR then
8                  Change the value to the closest upper or lower value with a probability of
                    0.5 each
9              else
10                 Assign a random value to the variable
11         else
12             Assign a random value to the variable
13     if The new solution is better than the worst solution in HM then
14         Replace the worst solution with the new solution in HM
```

2.2 Constraint Satisfaction Problems (CSP)

Many problems in the field of Artificial Intelligence can be modeled as Constraint Satisfaction Problems (CSP) which are in fact problems where a set of objects (variables) should be assigned to a state (values) that satisfies a set of constraints. Once a problem is modeled as a CSP, different techniques can be used in order to find solutions to the problem, as for example ACO, GA or HS.

As described in [62, 148], a **Constraint Satisfaction Problem** (CSP) is a problem that can be defined with three sets (X, D, C). The first set X contains the *variables* of the problem, $X = \{X_1, X_2, \ldots, X_n\}$. The set $D = \{D_1, D_2, \ldots, D_n\}$ contains the *domains* of the variables; for each variable X_i, all the possible values that can be assigned to X_i belong to its domain D_i. A finite set of *constraints* $C = \{C_1, C_2, \ldots, C_m\}$ limits the possible combinations of values that can be assigned to the variables.

Definition 2.2.1. An *assignment* A for a CSP is the allocation of a value x_i to variables $X_i \in X$.

$$A = \{(X_1, x_1), (X_2, x_2), \ldots, (X_m, x_m)\}$$

such that the value assigned to a variable belongs to its domain and any assigned variable has one single value assigned to it:

$$\forall (X_i, x_i) \in A : \quad x_i \in D_i$$
$$\forall (X_i, x_i), (X_j, x_j) \in A : \quad X_i = X_j \Rightarrow x_i = x_j$$

If there are variables in X that have no value assigned, the assignment is said to be *partial*. If values are assigned to all variables, the assignment is said to be *complete*. If all constraints $C_i \in C$ are satisfied by an assignment, the assignment is said to be *consistent*. If there is one or more constraints not satisfied, the assignment is said to be *inconsistent*. An assignment that is complete and consistent solves the CSP and is therefore a solution to the CSP.

2.2.1 Examples of CSPs

Different NP-hard problems can be modeled as CSPs [16]. A few classical examples are briefly described in this section.

2.2.1.1 The n-Queens Problem

The **n-Queens Problem** consists in placing n chess queens on a $n \times n$ chessboard so that no queen threatens another queen. There exist solutions for $n \geq 4$ [70]. A constraint satisfaction model of the problem has the three following sets:

- $X = (X_1, X_2, \ldots, X_n)$ has n variables, one for each queen, each queen being already assigned to a row in the chessboard, that is X_i is assigned to row i.
- The domain for each variable is the set of the columns in the chessboard, so:
  $D_i = \{1, \ldots, n\}$
- The constraints consist in forbidding the placement of two queens such that they are threatening each other:
  - One row contains one and only one queen. This is the case since X_i belongs to row i.
  - One column contains one and only one queen:
    $$\forall i, j \in \{1, \ldots, n\} : i \neq j \Rightarrow X_i \neq X_j$$
  - Any diagonal (complete or short) contains no more than one queen:
    $$\forall i, j \in \{1, \ldots, n\} : |i - j| \neq |X_i - X_j|$$
    where $|\cdot|$ represents the absolute value.

Figure 2.6 shows a possible solution to the 4-queens problem, where the sets of variables, domains and constraints of the CSP are:

- Variables: $X = \{X_1, X_2, X_3, X_4\}$.
- Domains: $D_1 = D_2 = D_3 = D_4 = \{1, 2, 3, 4\}$

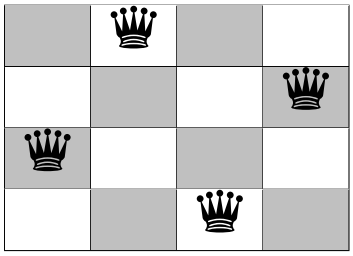

Figure 2.6: Solution to the 4-queens problem.

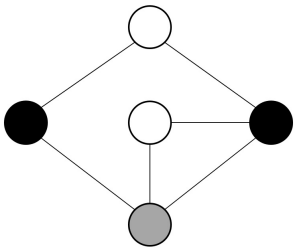

Figure 2.7: Solution to a Graph-Coloring problem with 5 vertices and 3 colors.

- Constraints:

$$\begin{array}{ccc} X_1 \neq X_2 & X_1 \neq X_3 & X_1 \neq X_4 \\ X_2 \neq X_3 & X_2 \neq X_4 & X_3 \neq X_4 \\ |X_1 - X_2| \neq 1 & |X_1 - X_3| \neq 2 & |X_1 - X_4| \neq 3 \\ |X_2 - X_3| \neq 1 & |X_2 - X_4| \neq 2 & |X_3 - X_4| \neq 1 \end{array}$$

2.2.1.2 The Graph-Coloring Problem

One variant of the **Graph-Coloring Problem** is based on a graph $G = (V, E)$ with n vertices V and m edges E:

$$V = \{v_1, v_2, \ldots, v_n\} \qquad E = \{(v_{i1}, v_{j1}), (v_{i2}, v_{j2}), \ldots, (v_{im}, v_{jm})\}$$

This problem consists in assigning a color to each vertex so that adjacent vertices do not share the same color. An example is given in figure 2.7

A constraint satisfaction model of the problem has the three following sets:

- $X = \{X_1, X_2, \cdots, X_n\}$ has n variables, one for each vertex.
- The domain for each variable is the set of available colors: $D_i = \{c_1, c_2, \cdots, c_k\}$, where k is the number of available colors.
- The constraints consist in forbidding to use the same color for two adjacent vertices:
  $\forall (v_i, v_j) \in E : X_i \neq X_j$

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
9	1	2	3	4	5	6	7	8
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
8	9	1	2	3	4	5	6	7
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4

Figure 2.8: Solution to a Sudoku 9×9 grid, with nine 3×3 sections in white/grey.

2.2.1.3 The Sudoku Game

Another example of CSP is the **Sudoku Game**. A classic Sudoku game with a 9×9 grid, as in figure 2.8, modeled as a CSP has the following sets:

- $X = \{X_1, X_2, \ldots, X_{81}\}$, where each variable X_i corresponds to one of the cells in the 9×9 grid.
- $D = \{D_1, D_2, \ldots, D_{81}\}$, where each domain D_i corresponds to the possible values of X_i, which means that $D_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}, \forall i \in \{1, 2, \ldots, 81\}$.
- $C = \{C_1, C_2, C_3\}$ where:
  - C_1: Any row of the grid contains all the values $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
    $\forall i \in \{1, 10, 19, 28, 37, 46, 55, 64, 73\} : Alldifferent(X_i, X_{i+1}, X_{i+2}, \ldots, X_{i+8})$
  - C_2: Any column of the grid contains all the values $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
    $\forall i \in \{1, 2, 3, \ldots, 9\} : Alldifferent(X_i, X_{i+9}, X_{i+18}, \ldots, X_{i+72})$
  - C_3: Any 3×3 section of the grid contains all the values $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
    $\forall i \in \{1, 4, 7, 28, 31, 34, 55, 58, 61\}$:
    $Alldifferent(X_i, X_{i+1}, X_{i+2}, X_{i+9}, X_{i+10}, X_{i+11}, X_{i+18}, X_{i+19}, X_{i+20})$

  where $Alldifferent(\cdot)$ means that all listed variables have a different value.

Figure 2.8 contains a solution to the 9×9 Sudoku problem. One value is assigned to each variable, so the assignment is complete. Every row, column and 3×3 section contains all values in $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, so the assignment is consistent.

The Sudoku game can also be modeled as a graph-coloring problem where each cell corresponds to a vertex. If two cells are in the same row, column or 3×3 section, then those cells are connected with an edge. Nine colors are available, each of them corresponding to a number in $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

2.2.2 Constraint Satisfaction Optimization Problems (CSOP)

In a CSP, all constraints are **hard constraints**, that means that all of them must be satisfied in order to have a consistent solution. Nevertheless, in most problems, another type of constraints has to be considered, the **soft constraints** that should be satisfied, but may be violated.

For example, when classes are scheduled in a school, there are hard constraints such as the fact that a teacher cannot teach two or more classes simultaneously. There are also soft constraints such as preferences of teachers who might prefer to teach in the afternoon. In different situations, a constraint may be soft or hard, a teacher can prefer to teach in the afternoon, which is a soft constraint, but another teacher is not available in the morning, which is a hard constraint.

The soft constraints are considered most of the time as a cost that should be optimized. This cost can be considered as the objective of the problem and is similar to the fitness function defined in Section 2.1.2.

As defined in [38, 148], a **Constraint Satisfaction Optimization Problem** (CSOP) is a CSP that has an objective to optimize. Those problems can thus be described with three sets and one function (X, D, C, f), where X, D and C are the set of variables, their domains and the set of constraints. f is a function that maps a numerical value to each possible solution.

As in a CSP, a solution to a CSOP is defined as an assignment of one value to each variable so that the value of each variable belongs to its domain and all the constraints of C are satisfied, as defined in Definition 2.2.1. The objective of the problem is not only to find one solution to the problem or the set of all possible solutions, but the goal in a CSOP is to optimize the value of f, that is to maximize or to minimize its value depending on the problem.

2.2.2.1 The Traveling Salesman Problem (TSP)

An example of a CSOP is the **Traveling Salesman Problem**, where there is a list of cities, a salesman and a travel cost associated to each pair of cities [4]. The problem consists in finding a sequence of those cities so that the salesman visits all cities and the total cost is minimized. The function f is thus the sum of the travel cost between the neighboring cities in the sequence.

The TSP can be modeled as a CSOP with the following sets:

- $X = \{X_1, X_2, \ldots, X_n\}$, where each variable X_i corresponds to the city in position i of the sequence.
- $D = \{D_1, D_2, \ldots, D_n\}$, where each domain D_i corresponds to the possible values of X_i, which means that $D_i = \{1, 2, \cdots, n\}, \forall i \in \{1, 2, \ldots, n\}$. D_i is thus the list of cities.
- The set of constraints C forces to have $X_i \neq X_j, \forall i, j \in \{1, 2, \cdots, n\}$.

If $c(i, j)$ is the travel cost between cities i and j, then the fitness function f for any solution $\{(X_1, x_1), (X_2, x_2), \ldots, (X_n, x_n)\}$ is calculated as given in Expression 2.7. The objective of the

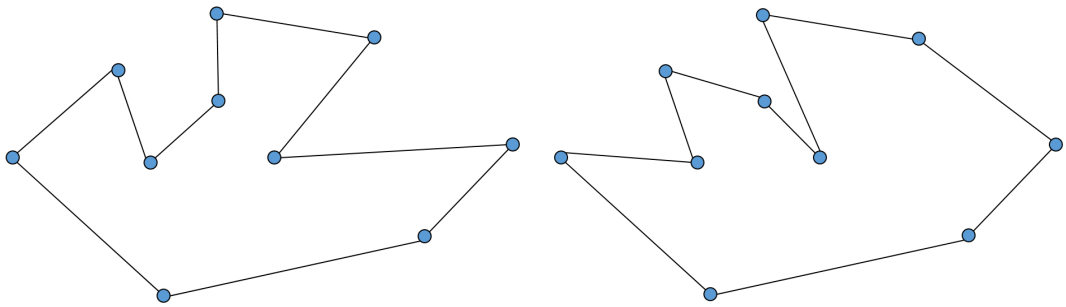

Figure 2.9: Two different solutions for the same TSP with 10 cities.

CSOP is to minimize the value of f.

$$f = \sum_{i=1}^{n} c(x_i, x_{i+1}) \tag{2.7}$$

Figure 2.9 represents an example of a TSP with 10 cities and two different solutions to visit all cities.

2.2.3 Search algorithms to solve CSPs and CSOPs

Different search algorithms can be used to solve CSPs and CSOPs, most of them are based on:

- **Depth-first search with backtracking** is a recursive algorithm where each step consists in selecting an unassigned variable and assigning a value to it. If all the constraints of C are satisfied afterwards, the partial solution is said to be consistent. If not, another value has to be tested. If no consistent value exists for this variable, then the algorithm goes back to the last assigned variable and assigns to it a value that has not been tested yet. The drawback of this method is that it might be time consuming for NP-hard problems whose size is large or even for mid-size problems.
- **Local search** algorithms consist in improving locally an assignment, by making small changes at each step, for example changing the value assigned to one variable. The drawback of this method is that there is no guarantee that the search does not fall in a local minimum in case of a CSOP.
- **Stochastic search** methods are used to speed up the search process. As many real problems are NP-hard, an exact approach or a local search are not efficient. A stochastic search consists in looking for solutions in a random way, that may be guided through some rules as in ACO. The drawback of this method is that there is no guarantee that the search finds the optimal solution and it is usually very difficult to know how far the found solution is from the optimal one.

There are several solvers that are available to model and solve CSPs and CSOPs such as Gecode, Choco or OR-Tools. **Gecode** is a software library developed in C++ that can be used

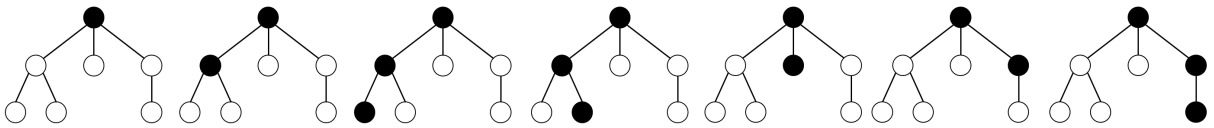

Figure 2.10: Steps in a Depth-first search with backtracking.

to solve both types of problems [131]. It contains all the tools needed to model a CSP: variables and their domain, and constraints. It also has engines that can be used to find solutions to the problem. As it is an open system, it allows users to implement their own variables, constraint definitions and search engines.

A constraint is implemented through a propagator that will remove from the domain of a variable all the values that are not compatible with this constraint. Gecode has already several standard propagators that are implemented, such as *distinct* that enforces integer variables to be all different, as the $Alldifferent$ constraint in the Sudoku game in Section 2.2.1.3. Another example is the *linear* propagator that posts linear constraints like $\sum_{i=0}^{n-1} a_i \cdot x_i = c$.

Regarding the structure of the search, Gecode is provided with options that allow the user to decide which variable should be considered next and which value should be considered for this variable, for example the variable with the smallest domain or the smallest value.

Two search engines are implemented in Gecode:

- **Depth-first search (DFS) with backtracking** that will either return complete solutions or inform that no solution exists.
- **Branch-and-bound** that can find, if it exists, the best solution to a problem that has an objective function.

The branch-and-bound splits the search space into smaller spaces (branching) and calculates a bound for the fitness function in those smaller spaces. This bound allows then to prune part of the tree and to speed up the search.

Several problems have been modeled and solved with Gecode, some of them are presented in the user manual [131]. Another example can be found in [69] where Hellkvist and Sjöstedt propose a model to solve a course timetabling problem. Gecode was also used to solve a multi-robot path planning problem modeled as a CSP [125]. Between 2008 and 2012, Gecode finished first in all categories of the MiniZinc Challenge, which is a competition of CSP solvers that uses a variety of benchmarks.

2.2.4 Integration of ACO and CSP models

The exact methods presented in Section 2.2.3 that can be used to solve CSPs are not always efficient when dealing with problems with a huge search space, but those methods are well-adapted to hard constraints. The situation is different for ACO algorithms which are efficient with huge search space, but not always with hard constraints.

The idea of combining an ACO algorithm with a CSP model has been proposed by Solnon in 2002 in [135], and in 2000 in [134] for a permutation CSP, which is a CSP where the values assigned to the n variables must be a permutation of n known values and must satisfy a set of constraints.

Since then, a few approaches have been proposed to integrate both methods and have been tested on real problems.

- In [136], an ACO algorithm is used to solve a classic CSP: the car sequencing problem.
- In [78], the standard backtracking procedure is replaced by an ACO search and this method is applied to the car sequencing problem.
- In [40], some variables are assigned with an ACO algorithm and some others with a branch-and-bound or a depth-first procedure. This hybrid metaheuristic is then applied to the problem of balancing a bike sharing system. The implementation of those methods was done using Gecode.

Algorithm 4 represents the ACO algorithm used to solve a CSOP. For each variable $X_i \in X$ and each $x \in D_i$, a pheromone is associated to this assignment: $\tau(X_i, x)$. The initialization, line 1, consists in assigning an initial value to all pheromones as in Expression 2.8.

$$\forall X_i \in X, \forall x \in D_i : \qquad \tau(X_i, x) = \tau_{init} \tag{2.8}$$

At each iteration, every ant builds a solution assigning a value to all variables. The probability of selecting a value x for a variable X_i depends on the pheromone corresponding to the assignment (X_i, x) and on a heuristic contribution, the visibility $\eta(X_i, x)$. The visibility gives an indication about the interest of an assignment, for example in the TSP, two cities that are close should have a higher probability, so a higher visibility, to be consecutive in the sequence than two cities distant from one another.

The probability of selecting a value $x \in D_i$ for the variable $X_i \in X$ is given by Expression 2.9 where $\eta(X_i, x)$ is the visibility of the assignment (X_i, x), α is the weight of the pheromone and β is the weight of the visibility.

$$\forall X_i \in X, \forall x \in D_i : \qquad Prob(X_i, x) = \frac{\tau(X_i, x)^\alpha \cdot \eta(X_i, x)^\beta}{\sum_{d_i \in D} \tau(X_i, d_i)^\alpha \cdot \eta(X_i, d_i)^\beta} \tag{2.9}$$

At the end of the iteration, the pheromones evaporate from all assignments and the ant that found the best solution deposits pheromones on the assignments of its solution. The probability to select those assignments in a future iteration will thus be incremented. This increment $\Delta\tau(X_i, x)$ is proportional to the quality of the solution found μ, if the solution is better, $\Delta\tau(X_i, x)$ is bigger. Expression 2.10 may be used to calculate $\Delta\tau(X_i, x)$ when the objective is to minimize the value of the fitness function f where φ is a parameter of the ACO

Algorithm 4: The ACO algorithm for a CSOP

Input: Number of iterations, number of ants
Output: Solution μ

Initialise pheromones, visibility and probabilities
for $i = 1, \ldots, NbIterations$ **do**
  $\mu_{iter} = \emptyset$
  for $j = 1, \ldots, NbAnts$ **do**
    $\mu = \emptyset$
    for $k = 1, \ldots, n$ **do**
      Select a value $x \in D_k$ for X_k with a probability $Prob(X_k, x)$
      $\mu = \mu \cup (X_k, x)$
    Calculate $f(\mu)$
    `// The best solution of the iteration is updated if the solution found by the ant is better`
    if $f(\mu)$ *is better than* $f(\mu_{iter})$ **then**
      $\mu_{iter} = \mu$
  `// The global best solution is updated if the solution of the iteration is better`
  if $f(\mu_{iter})$ *is better than* $f(\mu_{best})$ **then**
    $\mu_{best} = \mu_{iter}$
  `// The pheromones and the probabilities are updated`
  for $k = 1, \ldots, n$ **do**
    for $x \in D_k$ **do**
      $\tau(X_k, x) = (1 - \rho) \cdot \tau(X_k, x) + \Delta\tau(X_i, x)$
  Calculate the probabilities $Prob(X_i, x), \forall i \in \{1, \ldots, n\}$ and $x \in D_i$

algorithm.

$$\forall X_i \in X, \forall x \in D_i : \qquad \Delta\tau(X_i, x) = \begin{cases} \frac{\varphi}{f(\mu)} & \text{if } (X_i, x) \text{ belongs to the solution } \mu \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

2.3 Application fields

2.3.1 The Resource Allocation Problem (RAP)

The **Resource Allocation Problem** (RAP) consists in assigning a limited number of resources to objects (or agents). For example, we have RAPs in the following situations:

- In production, when tasks are assigned to people in a scheduling problem, the tasks are the objects and the people are the resources [68, 95].

- In asset management, the resource is the amount of money that can be invested and the objects are the different available assets [47].
- In universities, the students have to be allocated to courses, they are the objects and the seats in the different courses are the resources [17, 106].

A RAP can be modeled as a CSP where the variables correspond to the allocation of resources to agents, the domains contain the available resources and the constraints ensure an assignment that can be used as a solution to the problem [106]. Another option, more flexible, is that the variable X_{ij} represents if resource j is allocated to agent i, in this case, the domain of all variables is $\{0, 1\}$. A variable $X_{ij} = 1$ means that resource j is allocated to agent i, if $X_{ij} = 0$ then j is not allocated to i.

A detailed example can be found in Section 4.5 where the course allocation problem presented in Section 2.3.1.1 is modeled as a CSOP. The variables represent the courses (resources) allocated to students (agents). The domain of each variable is thus the list of courses offered. The constraints limit the number of students per course and the combinations of courses that can be allocated to a student.

Most of the RAPs include hard constraints that have to be considered. If any of those constraints is not fully satisfied in a solution, this solution is not consistent, as described in Definition 2.2.1. We can find for example capacity constraints as in the course allocation problem mentioned above [67, 17, 106]. Skills constraints are also important, for example when human resources are allocated to tasks, such as when nurses are allocated to shifts in [95].

Usually a RAP has an objective such as maximizing satisfaction or productivity, in which case, the RAP can be modeled as a CSOP: besides the sets of variables, domains and constraints, there is an objective function. In the three examples presented above, possible objectives for those RAPs are:

- In production: Maximize productivity.
- In asset management: Maximize revenue or Minimize risk.
- In universities: Maximize students' satisfaction.

If the objective is influenced by both sides, objects and resources, the RAP is often referred to as a **matching problem** [139]. For example, when seats are allocated to students in a university, students may submit preferences for the courses, in which case, the simple objective might be to maximize students' satisfaction [106]. If professors have also preferences for students, because of their background or their grades, then both types of preferences, students' and professors', have to be considered in order to find a good solution to the problem.

A classic example of matching problems is the stable marriage problem. In this problem there are two sets of elements of the same size, similar to a set of resources and a set of agents. Each element of a set has a ranking of preferences for the elements of the other set. A matching consists in mapping each element of one set to one element of the other set.

When comparing two solutions of a matching problem, there are two important characteristics: stability and Pareto-efficiency. A matching is said to be *stable* if there is no element A

in the first set and B in the second set such that A prefers B over the element A is currently mapped to, and B prefers A over the element B is currently mapped to. A matching μ is said to be *Pareto-efficient* if there is no other new matching such that in the new matching, all elements are at least as satisfied as in the first matching and at least one element is more satisfied than in the first matching.

Two examples of RAP are presented in the following sections: the first describes the Course Allocation Problem and the second the Course Timetabling Problem. In Section 2.3.2, the RAP is generalized to a many-objective RAP where several objectives have to be considered simultaneously.

2.3.1.1 Example of RAP: The Course Allocation Problem

The **Course Allocation Problem** (CAP) is critical in many universities and business schools. When students can select the courses they would like to attend, they often submit preferences, but have no guarantee that they will get a seat in all the courses they want to enroll in due mainly to quality and security constraints, that limit the number of seats available in a course [1, 21, 22].

Students may submit their preferences through different systems and seats are allocated with mechanisms often designed to maximize students' satisfaction [18, 41, 82, 106, 124, 138]. Two examples of those systems are:

- A **Ranking** system, where students rank the courses from their first choice to their last. Depending on the institutions, one or several courses may be ranked as a first choice. A formal definition of a ranking is presented below.
- A **Bidding** system that enables students to indicate how much they prefer a course over the others. Each student is credited with a specific amount of points and he bids points on his favorite courses. The number of points given to the different courses depends on the strategy of the student and on his preferences.

Definition 2.3.1. Let's consider S as the list of students and C as the list of courses. The *ranking* of a student $s \in S$ is a relation $\succeq_s$, such that $\forall c_i, c_j \in C$:

$$\begin{array}{lcl} c_i \succ_s c_j & \Leftrightarrow & s \text{ strictly prefers } c_i \text{ over } c_j \\ c_i =_s c_j & \Leftrightarrow & s \text{ has no preference of } c_i \text{ over } c_j \text{ or } c_j \text{ over } c_i \end{array}$$

With both systems, several allocation methods can be used. The simplest method is based on a First-come First-served approach. The first student to submit preferences is immediately allocated, as all courses are empty, this student receives his first choices. This process is applied to all students submitting preferences. This mechanism is very simple, but its main drawback is the unfairness for students: the last student to submit preferences is treated much worse than the first student.

Budish and Cantillon compare two mechanisms that use the Ranking system [17]. Both methods are applied once all students have submitted their preferences. The first mechanism, used in Harvard Business School, consists in allocating one course to each student at each step

of the process. For the first step the priority order of students is random and this order is reversed in each subsequent step. The second mechanism is the Random Serial Dictatorship which considers a random order of students and allocates all courses to one student at each step; it is a First-come First-served mechanism where the arrival sequence is defined by the random order.

Sönmez & Ünver analyze mechanisms that use the Bidding system [138]. The first one is similar to what is used at the University of Michigan Business School, which uses Bidding information to infer students' preferences. In this approach, the first choice of a student is the course on which he bids the highest number of points. Their conclusions show that this method results in an efficiency loss since there might be situations where a student bids more on a popular course and consequently is not assigned to his preferred course. From the student's point of view, this bidding is the only way to get this popular course, since a lot of students probably bid more points on it. This strategy leads to a situation where the student bids too few points to his favorite course and is therefore not assigned to it.

Diebold & al. analyze four matching methods to allocate a course to each student using three different metrics to compare the results [41]. In this problem, both the students and the courses have preferences over each other, which means that there is a ranking of the courses from the students and a ranking of the students from the courses. This second ranking might be due to the fact that professors prefer students with good grades.

Among the matching methods, Diebold & al. compare the *First-come First-served* (FCFS) and the *Random Serial Dictatorship* (RSD).

The *Gale-Shapley Student-Optimal Stable Mechanism* (SOSM) is a deferred acceptance algorithm. At each step, each student who still has to be allocated applies for his favorite course among the courses he hasn't applied to in the previous steps. Each course c has a capactiy q_c and selects a maximum of q_c students among the ones who applied for it in this step and who have been tentatively assigned to it in the previous one. Those students are tentatively assigned to this course, the others are rejected. The algorithm stops when no student is rejected.

The SOSM is represented in algorithm 5. $L_c(s)$ is the list of courses from which student s has not been rejected. $L_s(c)$ is the list of students who applied for c and have not been rejected. S_r is the list of students who have to apply to a course because they have been rejected from the ones they have already applied.

The SOSM is a mechanism that produces stable matchings, but that might be not Pareto-efficient.

The fourth matching method proposed by Diebold & al. is the *Efficiency Adjusted Deferred Acceptance Mechanism* (EADAM) and uses SOSM. EADAM runs SOSM and at the end of SOSM, students tentatively assigned to a course and rejected during the last step of SOSM, are identified. The course they were rejected from in this last step is removed from their preferences and SOSM is run again with those updated preferences. SOSM is run iteratively removing each time preferences from students. It stops when no course is removed from the students' preferences.

The first comparison metric is the *average rank*, that measures the global welfare of students. Each course has a rank for each student that is the position of this course in the student's ranking. The average rank calculates the average of the ranks of the courses allocated to students.

Algorithm 5: The Gale-Shapley Student-Optimal Stable Mechanism

Input: List of courses C, List of students S, Capacity q_c of each course $c \in C$, Ranking of courses by students $\succeq_s$, Ranking of students by courses $\succeq_c$

Output: List of students $L_s(c)$ assigned to each course $c \in C$

for $s \in S$ **do**
  $L_c(s) = C$
for $c \in C$ **do**
  $L_s(c) = \emptyset$
$S_r = S$
while $S_r \neq \emptyset$ **do**
  for $s \in S_r$ **do**
    `// Each student who has been rejected applies for his favorite course among the ones from which he has not been rejected`
    $c = \{c \in L_c(s) : c \succeq_s c_i, \forall c_i \in L_c(s)\}$
    $L_s(c) = L_s(c) \cup \{s\}$
    $L_c(s) = L_c(s) \backslash \{c\}$
    $S_r = S_r \backslash \{s\}$
  for $c \in C$ **do**
    `// Each overbooked course rejects the less favorite students among the ones who applied for it`
    while $Card(L_s(c)) > q_c$ **do**
      $s = \{s \in L_s(c) : s \preceq_c s_i, \forall s_i \in L_s(c)\}$
      $L_s(c) = L_s(c) \backslash \{s\}$
      $S_r = S_r \cup \{s\}$

The second comparison metric is the *popularity*, that compares matchings considering the number of students who prefer a matching over the other. A matching is more popular than a second matching if more students prefer the first matching to the second one.

Finally, the *rank distribution* compares how many students were assigned to their first choice, how many to their second choice, and so on.

Their conclusion is that SOSM and EADAM provide solutions fairer for students than FCFS or RSD since there is no order of students that defines a priority. The EADAM approach improves the global satisfaction of students compared to the SOSM.

2.3.1.2 Example of RAP: The Course Timetabling Problem

The **Course Timetabling problem** (CTT) has been widely studied for several decades. Nevertheless as every educational institution has its own requirements, the real problems that have to be solved are almost always different since they include constraints specific to one or several institutions [115]. The main components used on a CTT are resources (teachers, rooms, timeslots and students), activities (lectures, meetings, laboratory sessions, etc.) and constraints: Resources have to be assigned to activities and this assignment must satisfy the constraints.

Usually activities have to take place within 5 weekdays. This weekly schedule is then applied during a specific period of time, e.g. one semester or one year.

For schools, solutions have been proposed for specific countries or even specific schools [9, 117]. As each school has its own set of constraints, it is sometimes difficult or impossible to apply the same method that was used to solve a problem in a different school with different rules.

For universities, the CTT problem has been more studied than for schools. This is probably partly due to the fact that benchmark datasets have been provided through an international competition [103, 111]. Therefore two main variants of the CTT have been widely studied. In the Post-enrolment based variant (PE-TT), students have to select the courses they want to attend before the timetable is constructed. In the Curriculum-based variant (CB-TT), the university proposes several curricula, each of them containing a set of courses; to avoid clashes the courses of one curriculum should not overlap. In the first variant, PE-TT, the allocation of students to the courses is a constraint and clashes for students have to be avoided. In the second variant, a student selects a curriculum, and clashes are not admitted in a curriculum, so students have no clashes in their timetable [39, 90, 100].

The CTT problem can be modeled as a Constraint Satisfaction Problem (CSP), but as it is NP-hard, the search space is usually huge. The exact algorithms that have been developed to solve CTT problems can be classified into three main categories: Tree Search, like Branch-and-Bound; Dynamic Programming and Integer Programming. Due to the complexity of the problem, exact algorithms work properly only with small size problems. Therefore, from the first years of research in this field, metaheuristics have often been used to solve them [6, 89]. Evolutionary Algorithms and Tabu Search are the most popular methods [9, 89, 115, 121, 129]. Other techniques have also been used for example ACO [110, 132, 133] or PSO [143]. The drawback of some of those methods is that they are not always efficient dealing with hard constraints.

2.3.2 The Many-objective Resource Allocation Problem (MORAP)

In most real optimization problems, there are rarely situations where a single objective has to be considered in order to find a good solution to the problem. For example, when a family is looking for a house, it would ideally be close to work, close to school, big and cheap. Usually such objectives are not compatible and the solution that is selected, in the end, has to meet a trade-off between all objectives.

In a single objective optimization problem, there is often one single optimal solution. When dealing with several objectives, there are usually several solutions, whose set is called the Pareto front. A solution in the **Pareto front** is said to be non-dominated since it is not possible to improve one objective in it without penalizing one or more of the other objectives. All solutions in the Pareto front are equally good since there is no way to compare them considering only the objectives of the problem. Usually the objectives are opposite, as in asset management, maximize revenue and minimize risk [47], or in staff scheduling, minimize operational cost and improve working conditions [95].

In some situations, the objectives can be combined into a weighted function so that the problem becomes a single-objective RAP. When there are two or three objectives, the problem

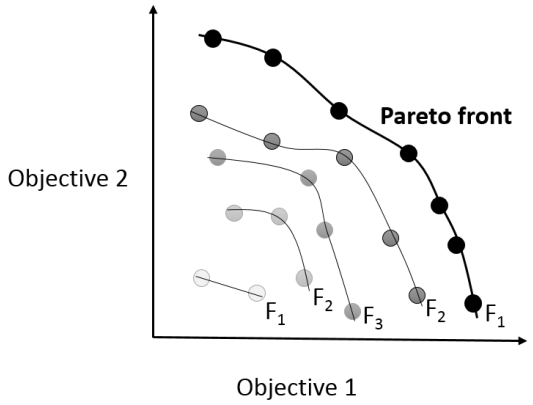

Figure 2.11: Example of a Pareto front (F_1) and of the successive Pareto fronts ($F_2, F_3, \ldots$).

is referred to as a **multi-objective problem** (MORAP) and the Pareto front can be easily visualized in a chart. When there are four or more objectives, the problem is called a many-objective problem and the visualization of the Pareto front becomes impossible if all objectives must be depicted.

Figure 2.11 illustrates a two-objective Pareto front, where both objectives have to be maximized. Each circle corresponds to a solution. The grey ones are dominated: there is at least one black point that is better for one of the objectives and at least as good for the other objective. The black ones are in the Pareto front, there is no solution that dominates them. This figure illustrates also how the solutions can be grouped into successive Pareto fronts. F_1 is the first Pareto front and contains all non-dominated solutions; F_2 is the second Pareto front and contains the solutions that are dominated only by solutions that belong to F_1; and so on.

Many approaches have been proposed in the past to solve multi-objective or many-objective optimization problems (MOOP), with Evolutionary Algorithms among the most popular techniques [24, 32, 57, 75, 156]. Other approaches have also been used such as PSO [25, 49], CRO [128], Simulated Annealing [3], HS [101, 127], or ACO [2, 120], among others.

When it comes to evolutionary algorithms, the well-known Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [33] has become a standard approach and is often used as a benchmark against which the performance of other algorithms is assessed [114, 48]. The NSGA-II is based on the identification of successive Pareto fronts, each of them being dominated by the previous ones. Those successive fronts along with a second criterion focused on the maximization of the crowding distance enable the selection of the best solutions that are used to continue the search [50].

In the recent years, different problems have been modeled as MORAP and solved with different types of approaches. Dynamic programming has been used to optimize a portfolio in a project financing firm [8]. Neighborhood search [91], PSO [49] and GA [113] have been applied to the classical problem of assigning workers to jobs where the objectives are to maximize the efficiency and to minimize the cost. Evolutionary algorithms have also been used for the allocation of projects to students where five different objectives had to be considered [119] and for the allocation of resources to projects over several periods of time [57].

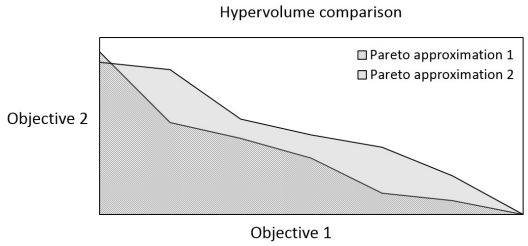

Figure 2.12: Two approximations of the Pareto set.

A common example can be found in finance, where investment portfolios are most of the time optimized considering return and risk. Usually the objective of a portfolio is to maximize the return, that is the expected revenue, and at the same time to minimize the risk. Nevertheless, when a high return is expected, the risk level is normally also high, and with a low risk level, the expected return is also low. There is no solution in the Pareto front better than the other solutions in the Pareto front. The selection of a non-dominated solution is based on other criteria such as the risk aversion of the investors.

2.3.2.1 The Hypervolume Metric

The hypervolume metric is an indicator often used to compare the performance of different approaches applied to many-objective optimization problems [73, 116, 156]. In a maximization problem, the hypervolume between the origin and the optimal Pareto front is maximized if all points of the optimal Pareto front are considered. When two approximations of the Pareto front are compared and one of them covers a greater hypervolume than the other, the latter is dominated.

Figure 2.12 contains an example with two approximations of the Pareto front of a MORAP with two objectives. For a MORAP with two objectives, the hypervolume corresponds to the surface between the origin and the Pareto front. The hypervolume of the first approximation covers the dashed area. The hypervolume of the second approximation covers the grey surface. Even if one solution of the dashed Pareto front is not dominated by the grey Pareto front, the grey hypervolume is greater than the dashed hypervolume. The hypervolume metric indicates that the grey approximation is better than the dashed approximation.

Monte Carlo simulation can be used when the computation of the hypervolume is too complex. A simple hypercube has to be considered, which contains all the non-dominated solutions, and whose hypervolume can easily be computed. In figure 2.12, the hypercube could be the rectangle that contains all non-dominated solutions. A number N of random points are then generated randomly in this hypercube. Among those N points, n are dominated by the approximation of the Pareto set. The hypervolume can thus be approximated by $\frac{n}{N} \cdot HV$, where HV is the hypervolume of the hypercube.

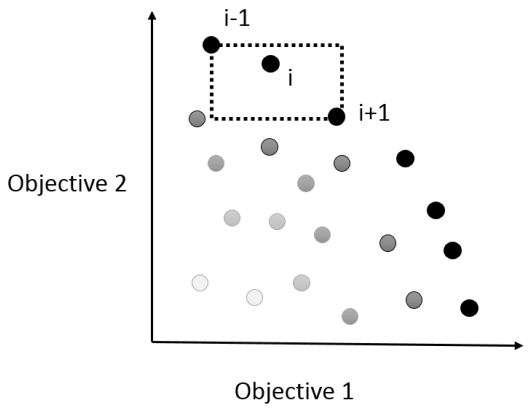

Figure 2.13: The crowding distance.

2.3.2.2 The Crowding Distance

When two solutions belong to two different successive Pareto fronts, as illustrated in Figure 2.11, both solutions can be compared since one of them dominates the other. When two solutions belong to the same front, the crowding distance enables this comparison . The crowding distance of a solution provides an estimation of the density of solutions around this solution. It is computed considering its neighbors for each objective in the front the solution belongs to.

One example for the computation of the crowding distance is given here. Let F_i be one of the successive fronts. For one objective and its fitness function f, let $V = \{v_1, \ldots, v_k\}$ be all the distinct values taken by f in the solutions belonging to F_i, those values are sorted in ascending order. For a solution $\mu \in F_i$, if $f(\mu) = v_i$, its crowding distance for f is given by Expression 2.11 and its global crowding distance is the sum of the crowding distances for all objectives as given by Expression 2.12. Figure 2.13 illustrates the crowding distance of a solution i.

$$CD_f(\mu) = \begin{cases} 1 & \text{if } i \in \{1, k\} \\ \dfrac{v_{i+1} - v_{i-1}}{v_k - v_1} & \text{otherwise} \end{cases} \tag{2.11}$$

$$CD(\mu) = \sum_f CD_f(\mu) \tag{2.12}$$

2.3.3 The Vehicle Routing Problem (VRP)

The Vehicle Routing Problem (VRP) was first presented formally by Dantzig and Ramser in 1959 as a problem of gasoline delivery from a bulk terminal to a large number of service stations [29]. The VRP consists in finding routes to serve a given number of customers from one or several depots, that is to collect or to deliver goods to those customers [29, 85, 147]. Figure 2.14 represents a VRP with 3 depots (houses) and 14 customers (circles). All customers are served from one depot since each of them belongs to one of the 4 routes.

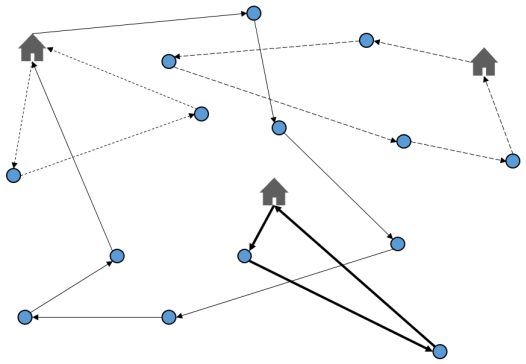

Figure 2.14: A VRP problem with 3 depots, 14 customers and 4 routes.

The VRP is a generalization of the TSP, presented in Section 2.2.2. A simple definition of a simple VRP can be made with a graph $G = (V, E, C)$ where $V = \{v_0, v_1, v_2, \ldots, v_n\}$ is the set of vertices, $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of arcs and $C = \{c_{ij}\}$ is a cost matrix that can represent distances, travel times or costs associated to each arc of E. A VRP consists in finding a set of routes that all start and finish at the vertex v_0, the depot, and visit all other vertices $\{v_1, v_2, \ldots, v_n\}$ exactly once, considering different types of constraints [19]. The objective is to minimize the total cost of those routes.

Several variants of the VRP have been studied, each of them with different types of constraints, among them:

- VRP with Time Windows (VRPTW). Each customer has a time window and has to be served within this time window. A customer has an earliest service time, it cannot be served before, and a latest service time, it cannot be served after this deadline [137, 150].
- Capacitated VRP (CVRP). The vehicles have a limited capacity and the goods loaded on a vehicle cannot exceed this capacity [51, 79]. The fleet in this variant can be homogeneous, all vehicles have the same characteristics [29], or heterogeneous, vehicles have different capacities or different time characteristics regarding for example availability [60, 42, 98].
- VRP with Multiple Trips (VRPMT). A vehicle can do more than one route, it can have a sequence of routes, each of them serving a sequence of customers.
- VRP with Simultaneous Pickup and Delivery (VRPSPD). Serving a customer means to deliver goods to him or to pick up goods from him or both simultaneously [37, 34].

Those variants are often combined since in real-life problems, several constraints' types have to be considered, as a heterogeneous fleet for pickup and delivery with time windows at the customers [5, 11].

In a more generalized version of VRP, at each vertex, there is a customer with a given demand for delivery or for pickup, a depot or a fleet of vehicles. The objective is to find a set of routes that visit all or a maximum of customers and minimize the total cost. A vehicle can perform one or more routes and each route starts at a vertex where a vehicle is available and

finishes at a vertex that corresponds to a parking lot. In addition, there is a set of constraints that have to be satisfied in order to have a feasible solution to the problem. Those constraints consider for example:

- Time: time windows or loading or unloading time at the customer's facility and depots, time availability of the vehicles, maximum working and driving times [83];
- Compatibility: between vehicle and customer, between customer and depot, between customers served in the same route [36];
- Capacity: of vehicles, of depots.

Regarding the cost, in addition to the travel cost for each edge used by the routes, several costs have to be considered most of the time. For example, there can be a cost associated with the working time of the routes or a fixed cost generated per use of a vehicle, in which case it might be interesting to have more travel costs, but fewer vehicles. All those costs can also vary depending on the time of day [66]. There might even be a cost penalty due to delivery delay [35]. The objective of a VRP is usually to minimize the total cost of the solution.

Many different approaches have been used to solve VRPs. Some of them are exact methods such as Branch-and-Bound [86, 146], or Dynamic Programming [83], but those exact methods become inefficient when the size of the problem grows as VRPs are NP-hard problems [87]. Therefore, most of the approaches proposed in the literature are approximate methods applied to a given variant of the VRP. Part of those approaches can be roughly classified as metaheuristics or heuristics.

PSO is an example of a metaheuristic that has been used to solve different variants such as VRP with simultaneous pick-up and delivery or with time windows [11, 59, 72, 97]. Another metaheuristic is ACO that has been applied to several variants of VRPs [10], for example with multi-compartment vehicles [123], or with time windows with uncertain travel times [145]. GA have also often been used to solve VRPs [7, 77, 112, 118]. Tabu Search is another metaheuristic used to solve different variants of the VRP, among them the simple one and the multi-depot variant, with and without time windows [27].

Chapter 3

Heuristic and ACO Approaches for Resource Allocation Problems

This chapter presents the problem of allocation of students to classes in secondary schools in Switzerland where students are grouped in main classes and split and grouped differently for some subjects. In the first part, a complete constraint-based model is proposed for the problem that combines the class allocation problem with a scheduling problem that considers the different classes and how the schedule of the students has to be organized in order to minimize the number of lessons and of teachers.

In the second part, a model and two resolution approaches are presented for the allocation of students to classes in order to have classes as mixed as possible regarding the profile of students. This allocation problem is first modeled as a CSOP. The CSOP is then solved with a CSOP solver and with an ACO algorithm. Eight real datasets from schools in Canton de Vaud are used to compare the performance of the two approaches.

The last section deals with a special case of the Course Timetabling Problem. This special case has been developed as a benchmark and is very often used to compare the performance of different algorithms proposed to solve timetabling problems. The benchmark datasets have been used in an international competition in 2007, but are currently still used in the field. The approach presented here is based on ACO hybridized with a local search.

3.1 Introduction

This chapter is based on the Course Timetabling (CTT) problem faced by secondary schools in Canton de Vaud, a French-speaking part of Switzerland. A new educational system was launched in 2013, [30, 104]. Before 2013, students were allocated to classes and then a traditional timetabling problem was solved with a commercial software in almost all schools in Canton de Vaud, since all students in a class attended the same lessons. In the new system, there still exist classes, but a student will be part of one main class and of several other classes depending on the lessons he has to attend. Students in the same main class are therefore split and grouped differently with other students for some activities scheduled during the week. This splitting-

grouping process is not specific to Switzerland, it exists in high schools in Greece [9], Germany [74] and Australia [80]. This process allows to have students in the same main class who are split for some subjects as for elective courses or because of their level in the subject. In Switzerland, those subjects represent sometimes more than half of the timetable of a student.

In most cases, the splitting-grouping of students has an impact on the timetable, but is not considered as an objective. In Canton de Vaud, a student can be described with a profile with the options he selected and his level in the different fields. The pedagogical objective is to have main classes as mixed as possible and to have a similar diversity between classes. Mixed classes are preferred in order to avoid situations where students with the same profile are allocated to the same class. Students can thus be seen as resources allocated to main classes, and the objective of the RAP is the diversity within each class and the similarity between classes. Regarding diversity, classes are competitors since an assignment may lead to one class perfectly mixed penalizing the others. Regarding similarity, classes have to distribute students in a fair way among them in order to satisfy this objective as much as possible.

In the first part of this chapter, we propose a complete constraint-based model for the Canton de Vaud problem that combines both a CTT and a RAP. In the second part, we present a model and two resolution approaches for the allocation of students to classes in order to have classes as mixed and similar as possible. This second problem is first modeled as a Constraint Satisfaction Optimization Problem (CSOP), [148]. The CSOP is then solved with a CSOP solver, Gecode [131], and with an ACO algorithm, [38, 44, 136].

The rest of the chapter is structured as follows. Section 3.2 describes the problem in Canton de Vaud. Sections 3.3 and 3.4 present the model for the combined problem CTT & RAP and its complexity. Section 3.5 describes the model for the simple RAP, the class allocation problem. Sections 3.6 and 3.7 describe the CSOP and the ACO approaches. Section 3.8 contains the description of the results. Section 3.9 contains the conclusions.

Appendix A contains a glossary of the notation used in this chapter.

3.2 Description of the problem

In Canton de Vaud, a new organization for the compulsory schooling has been introduced step by step since August 2013 for the new students; the previous structure was kept for the students who were already in secondary school in 2013. The secondary school includes three years of the Swiss educational program (grades 9 to 11, around years 12 to 15) and there are about 90 schools in Canton de Vaud.

Sections. Students are divided into two sections: *Voie Générale* (VG) and *Voie Prégymnasiale* (VP) depending on their grades. VP students are prepared to access a high school, VG students are prepared to access a vocational school.

Options. Each student assigned to VP has to select one specific option (OS) among four topics. Each student assigned to VG has to select two options (OCOM), one among two topics and one among six other topics. Students in VG may select an OS instead of two OCOMs if some criteria are satisfied, in which case those VG students attend the OS lessons with the VP students. Table 3.1 contains the list of the available options for both sections.

Section	Group	Topic
VP / OS		Economics and Law
		Latin
		Italian
		Math and Physics
VG / OCOM	1	French
		Math
	2	Creative and Manual Activities
		Visual Arts
		Economics, Law and Citizenship
		Nutrition Education
		Technology
		Natural Science

Table 3.1: Available Options for VP and VG.

Levels. VG students are also classified into levels for three important subjects: French, German and Mathematics (FGM), again depending on their grades. Levels 1 and 2 are part of the VG section, i.e. a class has only VG students; students with level 3 attend the subject with the VP students.

For each grade and each section, one or several classes are open in a school; each class may contain up to 24 students. All students of a class attend the standard courses together, i.e. courses with no options and no levels, such as English, Sports, etc. Students who selected an OS are split into groups and each student attends his own OS course together with the students of other classes who also selected the same OS. Nevertheless, due to capacity constraints, there might be several sessions for a popular OS, in that case, students who selected this OS are split among those sessions. The same rule applies for OCOMs and for levels in FGM subjects.

At the end of each semester, a student may be assigned to a different section (from VG to VP or from VP to VG) or to a different level for the three FGM subjects.

Considering the levels for FGM and the options OS and OCOM, for each grade in a school there might be up to 4 different profiles for VP students and up to 432 profiles for VG students in schools that propose 6 OCOMs. A school must propose at least 2 OCOMs out of the 6 available, and even with only 2 OCOMs, they have up to 216 different profiles.

Example. Let's consider two classes: Q1 and Q2 with 4 students each and their French level as given in Table 3.2. In that case, each session of the English course is assigned to a whole class. But students' allocation to French sessions depends on their levels. Session 1 contains only students with level F_1 and session 2 contains only students with level F_2. If capacity constraints limit the number of students per session to 4, there would be three French sessions since 5 students have level F_1.

In all schools, the director is in charge of assigning students to classes and of creating a timetable. For the timetabling problem, they use a software that was sufficient before the introduction of the new school organization in 2013. However since 2013, the timetabling problem is combined with the problem of allocating students to main classes. The directors currently solve

	Class 1				**Class 2**			
Student	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
French level	F_1	F_1	F_2	F_2	F_1	F_1	F_1	F_2

With no capacity constraints

	Session 1					**Session 2**			
English course	S_1	S_2	S_3	S_4		S_5	S_6	S_7	S_8
French course	S_1	S_2	S_5	S_6	S_7	S_3	S_4	S_8	

With capacity constraints: $\leq$ 4 students/session

	Session 1			**Session 2**		**Session 3**		
French course	S_1	S_2	S_5	S_6	S_7	S_3	S_4	S_8

Table 3.2: Example of allocation of students to sessions.

the problem in two steps: first, students are manually allocated to classes and then the timetable is created by running the software. However, due to options and levels, the way the students are allocated to classes may lead to unfeasible timetables because of resource constraints. Therefore it might be a long process since directors have to test different allocations. This is a problem since on one hand the students' results are known only at the end of the academic year, but on the other hand the teachers must know the courses they are going to teach before the end of the academic year and the director must have time to recruit new teachers if needed.

3.2.1 Objectives

There are two educational objectives to consider, each of them is described below. Their sequence indicates their priority: the first objective has the highest priority and the second the lowest.

Similar difficulty. One educational objective is to have the same difficulty level for all weekdays. There are some hard subjects, such as Mathematics, German, etc. and some lighter, such as Sports, Visual Arts, etc. One weekday should not contain only hard subjects while another contains only light subjects.

Mixed classes. Another educational objective is that the composition of one VG class should be as mixed as possible regarding the profile of the students in the FGM courses and the composition of different VG classes of the same grade should be as similar as possible. For example, if we consider the French course, 10 students with level F_1 and 10 students with level F_2, it is better to have 2 classes, each of them with 5 students of level F_1 and 5 students of level F_2, than to have 2 classes, one of them with 10 students of level F_1 and the other with 10 students of level F_2.

However the timetabling of mixed classes is much more constrained since the schedule of several classes must be aligned. In the mixed option of the previous example, both classes must have the same schedule for the French lessons since all students in F_1 must attend the same lessons (as do F_2 students). Moreover, two French teachers are needed since lessons for F_1 and for F_2 are simultaneous. If the school has only one French teacher, then classes cannot be mixed

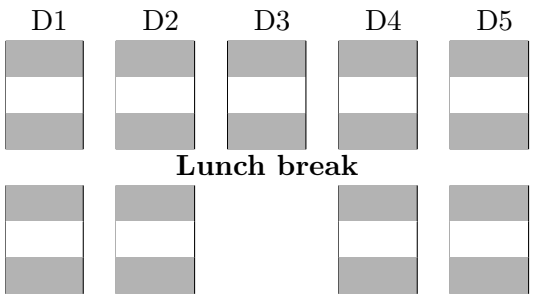

Figure 3.1: Canton de Vaud: Week structure: 5 days split into 2 half-days (except Wednesday) and 4 quarter-days (grey).

for French, therefore all F_1 students must be in the same class (as do all F_2 students) and the teacher teaches French in both classes at different times.

3.3 The complete model for the combined CTT & RAP

This section describes the proposed model in detail. Section 3.3.1 describes the different concepts that have to be considered in the problem of Canton de Vaud. Section 3.3.2 contains the results that correspond to a complete solution to the problem. Section 3.3.3 describes the hard constraints that have to be satisfied in order to have a feasible timetable and Section 3.3.4 explains how the soft constraints are considered as a cost for the optimization problem.

3.3.1 Concepts

One week is divided into different concepts, figure 3.1 represents the structure of one week in Canton de Vaud: **Day** ($d \in D$), **Half-day**, **Quarter-day** and **Period** ($p \in P$). Each day has up to 2 half-days separated by a lunch break. Each half-day has up to 2 quarter-days separated by a small break. Each quarter-day has up to 3 periods. A period p cannot be divided into smaller items; it belongs to day $PD(p)$ and may be the first or the last period of a half-day or quarter-day $PHF(p), PHL(p), PQF(p), PQL(p) \in \{0, 1\}$.

A period has a neighborhood $PN(p)$ of 1 or 2 periods. If $PQF(p) = 1$, then p is the first period of a quarter-day and $PN(p)$ contains the period that follows p. If $PQL(p) = 1$, then p is the last period of a quarter-day and $PN(p)$ contains the period that precedes p. If $PQF(p) = PQL(p) = 0$, then $PN(p)$ contains both periods, the successor and the predecessor of p.

Grade. In Canton de Vaud, there are 3 grades in secondary schools: 9, 10 and 11. For all grades, a student cannot have more than P_{max} periods per day. $P_{max} = 8$ in Canton de Vaud.

Subjects. A subject is an area of study, e.g. Mathematics, English, Visual Arts, etc.

Rooms, $r \in R$. The information needed for a room is if it requires travel time, $RT(r) \in \{0, 1\}$.

Teachers, $t \in T$. Each teacher has a maximal and a minimal number of periods he can work per week, $TW_{max}(t)$ and $TW_{min}(t)$, and a list of periods when he is available $TP(t)$.

Categories, $k \in K$. In the model proposed in this chapter, a category is a concept that implies a specific grouping of students. A grade-section is a category, e.g. *Grade 9-VP* and *Grade 10-VG* are categories. A grade-OS subject is also a category, e.g. *Grade 11-Latin*, as is a grade-OCOM subject, e.g. *Grade 10-Visual Arts*. Each level of subjects with levels is a category, e.g. *Grade 9-Math-Level 1*, *Grade 10-French-Level 3*.

A category is defined with a maximal number of students per class $KS_{max}(k)$, its grade $KG(k)$ and its type $KT(k) \in \{section, OS, OCOM, level\}$. For all types, except *section*, a category has a single subject. For every grade, there is only one category whose type is *section*.

Classes, $q \in Q$. A class is a grouping of students and is defined with a category $QK(q) \in K$. All students who belong to a category must be allocated to one class of this category. The number of classes of a category depends on the number of students belonging to this category.

Courses, $c \in C$. A course belongs to a class $CQ(c) \in Q$. A class has one or several courses depending on its category: A class of a grade-section category will include all courses with no levels and no options. The classes of the other categories' types have only one course.

A course has a list of adapted rooms $CR(c) \in R$, a list of skilled teachers $CT(c) \in T$, a maximum number of lessons per day $CD_{max}(c)$, and a weight $CW(c)$.

Some courses shouldn't be followed or preceded by others, each course has therefore non-desirable neighbors $CF(c)$, e.g. a German lesson right after an English lesson. On the other hand some courses should be scheduled just before or after others, each course has therefore desirable neighbors $CN(c)$, e.g. Visual Arts and Music should be scheduled in consecutive periods since students will have 2 periods fortnightly instead of 1 period per week for both courses.

Lesson, $l \in L$. A lesson is a teaching unit of a course that can be scheduled in one period. It is defined with its course $LC(l) \in C$. A course has therefore a list of lessons $\{l \in L : LC(l) = c\}$.

Students, $s \in S$. A student is defined with the categories he belongs to $SK(s)$. A student belongs to one single category whose type is *section*.

3.3.2 Solution

A solution to the complete CTT & RAP problem is described in Definition 3.3.1.

Definition 3.3.1. A *solution* $\mathcal{S} = \{\mu_k, \mu_T, \mu_R, \mu_P\}$ to the complete CTT & RAP is defined as the set of the assignments described below, where $k \in K, l \in L, t \in T, r \in R, p \in P$. There are $Card(K) + 3$ sets in $\mathcal{S}$ since there is one set per category $k \in K$.

μ_k:	$(s,q) \in \mu_k$	$\Leftrightarrow$	$q = \mu_k(s)$	is the class of category k allocated to student s Note: If $k \notin SK(s)$, then $\mu_k(s) = \emptyset$
μ_T:	$(l,t) \in \mu_T$	$\Leftrightarrow$	$t = \mu_T(l)$	is the teacher allocated to lesson l
μ_R:	$(l,r) \in \mu_R$	$\Leftrightarrow$	$r = \mu_R(l)$	is the room allocated to lesson l
μ_P:	$(l,p) \in \mu_P$	$\Leftrightarrow$	$p = \mu_P(l)$	is the period allocated to lesson l

The list of lessons a student has to attend can thus be deduced. A student has to attend the lessons whose course belongs to the one of the classes the student is assigned to:

$$\nu_L(s) = \{l \in L : LC(l) \in \mu_k(s) \wedge k \in SK(s)\} \tag{3.1}$$

3.3.3 Hard Constraints

The hard constraints in an optimization problem must always be satisfied in order to have a feasible solution. The hard constraints for the CTT&RAP are described below; they are classified into three categories according to their type. *Basic Constraints* exist in almost all timetabling problems. *Assignment Constraints* are related to the assignment of students to classes and to the assignment of lessons to periods. *Resources Constraints* refer to the use of the available resources.

- **Basic Constraints**
  - A lesson must be assigned to one period, one room and one teacher, $\forall l \in L$:
    $$(\mu_P(l) \in P) \wedge (\mu_R(l) \in R) \wedge (\mu_T(l) \in T)$$
  - A student cannot attend 2 lessons at the same time, $\forall s \in S, \forall l_1, l_2 \in L$:
    $$l_1, l_2 \in \nu_L(s) \Rightarrow \mu_P(l_1) \neq \mu_P(l_2)$$
  - A teacher cannot teach 2 lessons at the same time, $\forall l_1, l_2 \in L$:
    $$\mu_P(l_1) = \mu_P(l_2) \Rightarrow \mu_T(l_1) \neq \mu_T(l_2)$$
  - A room cannot host 2 lessons at the same time, $\forall l_1, l_2 \in L$:
    $$\mu_P(l_1) = \mu_P(l_2) \Rightarrow \mu_R(l_1) \neq \mu_R(l_2)$$
  - All lessons of a course must be assigned to the same teacher, $\forall l_1, l_2 \in L$:
    $$LC(l_1) = LC(l_2) \Rightarrow \mu_T(l_1) = \mu_T(l_2)$$
  - All lessons of a course must be assigned to the same room, $\forall l_1, l_2 \in L$:
    $$LC(l_1) = LC(l_2) \Rightarrow \mu_R(l_1) = \mu_R(l_2)$$
- **Assignment Constraints**
  - A student must be assigned to one class of each category he belongs to, $\forall s \in S, \forall k \in K$:
    $$k \in SK(s) \Rightarrow \mu_k(s) \in Q$$
  - A student cannot attend more than P_{max} lessons per day, $\forall s \in S, \forall d \in D$:
    $$Card(\{l \in \nu_L(s) : PD(\mu_P(l)) = d\}) \leq P_{max}$$

- A course cannot have more than $CD_{max}(c)$ lessons scheduled per day, $\forall c \in C, \forall d \in D$:

$$Card(\{l \in L : LC(l) = c \wedge PD(\mu_P(l)) = d\}) \leq CD_{max}(c)$$

- A student can have an idle period only if it is the first or the last period of a half-day, $\forall s \in S, \forall p \in P$:

$$p \notin \{\mu_P(l) \in P : l \in \nu_L(s)\} \Rightarrow (PHF(p) = 1) \vee (PHL(p) = 1)$$

- The lessons of courses that cannot be neighbors cannot be consecutive, $\forall l_1, l_2 \in L$:

$$LC(l_1) \in CF(LC(l_2)) \Rightarrow \mu_P(l_1) \notin PN(\mu_P(l_2))$$

- The lessons of neighboring courses must be consecutive, $\forall c_1, c_2 \in C$:

$$c_2 \in CN(c_1) \Rightarrow \exists l_1, l_2 \in L : (LC(l_1) = c_1) \wedge (LC(l_2) = c_2) \wedge (\mu_P(l_1) \in PN(\mu_P(l_2)))$$

- **Resources Constraints**

  - The number of lessons of a teacher is limited, $\forall t \in T$:

$$TW_{min}(t) \leq Card(\{l \in L : \mu_T(l) = t\}) \leq TW_{max}(t)$$

  - A teacher can teach only if he is available, $\forall l \in L$:

$$\mu_T(l) = t \Rightarrow \mu_P(l) \in TP(t)$$

  - A teacher can teach only if he is skilled in the course, $\forall l \in L$:

$$\mu_T(l) \in CT(LC(l))$$

  - Courses and rooms must be compatible, $\forall l \in L$:

$$\mu_R(l) \in CR(LC(l))$$

  - The number of students per class is limited, $\forall q \in Q$:

$$Card(\{s \in S : \mu_k(s) = q\}) \leq KS_{max}(QK(q))$$

  - Lessons scheduled in rooms that require travel time must be placed before or after a break, $\forall l \in L$:

$$RT(\mu_R(l)) = 1 \Rightarrow PQF(\mu_P(l)) = 1 \vee PQL(\mu_P(l)) = 1$$

3.3.4 Soft Constraints

Soft Constraints are constraints that should be satisfied as much as possible. Usually a timetable where all the soft constraints are satisfied does not exist. Therefore a cost is associated with a soft constraint that is not satisfied and the objective of the problem is to minimize the total cost of the solution. The two objectives described in Section 3.2.1 are the soft constraints of the problem.

Similar difficulty. The difficulty level of day d for student s is the sum of the difficulty level of each lesson that is on his schedule for that day; the difficulty level of a lesson being the weight of the corresponding course: $CW(LC(l))$. The list of lessons that s has to attend on d is

$$\nu_{LD}(s,d) = \{l \in \nu_L(s) : PD(\mu_P(l)) = d\} \tag{3.2}$$

Therefore, the weight of $d \in D$ for $s \in S$ is

$$\nu_W(s,d) = \sum_{l \in \nu_{LD}(s,d)} CW(LC(l)) \tag{3.3}$$

The difficulty level of all days should be between W_{min} and W_{max}, the total cost is defined by an overweight cost C_{OW} and an underweight cost C_{UW}

$$C_{UW} = \sum_{d \in D} \sum_{s \in S} (W_{min} - \nu_W(s,d))^+ \tag{3.4}$$

$$C_{OW} = \sum_{d \in D} \sum_{s \in S} (\nu_W(s,d) - W_{max})^+ \tag{3.5}$$

Mixed classes. Let's consider a class $q \in Q$ and a category $k \in K$. The number of students in q that belong to category k is

$$\nu_S(q,k) = Card(\{s \in S : \mu_k(s) = q\}) \tag{3.6}$$

Two classes $q_1, q_2 \in Q$ are similar for a category $k \in K$, if $\nu_S(q_1,k) = \nu_S(q_2,k)$. The classes of a section category k_m are perfectly mixed if they are all similar for any category corresponding to a level. Therefore we can define the following cost:

$$C_M = \sum_{k_m} \sum_{k_l} \sum_{q_1} \sum_{q_2} (\nu_S(q_1,k_l) - \nu_S(q_2,k_l))^+ \tag{3.7}$$

where

$$\begin{array}{lcl} k_m & \in & \{k \in K : KT(k_m) = section\} \\ k_l & \in & \{k \in K : KT(k) = level \wedge KG(k_l) = KG(k_m)\} \\ q_1 & \in & \{q \in Q : QK(q_1) = k_m\} \\ q_2 & \in & \{q \in Q : QK(q_2) = k_m \wedge q_2 \neq q_1\} \end{array}$$

Total cost. The total cost of a timetable and an assignment of students to classes is given by Expression 3.8.

$$Cost = \alpha \cdot C_{UW} + \beta \cdot C_{OW} + \gamma \cdot C_M \tag{3.8}$$

where α, β and γ are parameters that have to be tuned according to the objectives of the timetabling problem.

3.4 Complexity of the problem

The CTT problem is NP-hard, it is therefore not possible to find the optimal solution for any instance in a time that is reasonable for a school, [26]. In addition to that there might be

instances, i.e. schools, where the constraints are such that there is no feasible solution because the resources are scarce compared to the needs or where only a few feasible solutions exist, but may be quite difficult to find.

Regarding the allocation of students to classes, there is a big difference between situations where there are VG students that attend VP courses and situations with 2 independent sections that only share rooms and teachers, but not students.

The subjects with levels are also critical for the timetable. If students of different main classes have to be mixed and split for the French course, there must be enough teachers available to teach French simultaneously. Constraints related to teachers play also an important role here.

In Canton de Vaud a lot of teachers work part-time and the number of subjects a teacher is skilled in is limited. Therefore the constraints related to teachers should be considered first since they reduce significantly the search space. Regarding rooms, most schools have only one or two rooms adapted to a few specific subjects, e.g. Gym hall for Sports. Neighboring subjects imply that when one of them is planned, only 1 or 2 periods are possible for the other.

Considering this information, the subjects can be classified according to the difficulty of scheduling their lessons in the timetable:

1. Subjects that share VP and VG students
2. Subjects with levels in VG
3. Subjects with teachers with little availability
4. Subjects with a specific room
5. Subjects with neighbors
6. Other subjects

3.5 The model for the CLass Allocation Problem (CLAP)

This section describes the RAP model for the Class Allocation Problem (CLAP) in detail. The concepts of the CLAP are part of the concepts defined in Section 3.3:

- **Grade**, $g \in G$.
- **Categories**, $k \in K$.
- **Classes**, $q \in Q$. In this section, the classes that are considered are only the section classes, since the way the students are split in the other classes has no impact on the objectives.
- **Students**, $s \in S$.

Definition 3.5.1. A *solution* $\mu = \{(s,q) \in \{S \times Q\}\}$ to the class allocation problem contains the assignment of students to classes:

$$(s,q) \in \mu \Leftrightarrow \text{ Student } s \text{ is assigned to class } q \tag{3.9}$$

Definition 3.5.2. A solution is said to be *feasible* if the following hard constraints are satisfied:

1. A student must be assigned to one class of each category he belongs to. Expression 3.10 states that if a student belongs to a category, there exists a unique class of this category to which the student is allocated. Expression 3.11 says that if a student is allocated to a class, the this student belongs to the category of this class.

$$\forall s \in S, \forall k \in SK(s) \quad : \quad \exists!(s,q) \in \mu : QK(q) = k \tag{3.10}$$
$$\forall (s,q) \in \mu \quad : \quad QK(q) \in SK(s) \tag{3.11}$$

2. The number of students per class is limited and depends on its category as described in Expression 3.12

$$\forall q \in Q \quad : \quad Card(\{s \in S : (s,q) \in \mu\}) \leq KS_{max}(QK(q)) \tag{3.12}$$

In the CLAP, the objective is to have diversity within section classes and similarity between them. The similarity objective contains implicitly the diversity objective. Indeed if all classes are similar, then they are equally diverse.

For an allocation μ, the number of students in a class q that belong to a category k is given by Expression 3.13. The average number of students of a category $k \in K$ allocated to a section class should be the number of students belonging to k divided by the number of section classes of the grade of k, as in Expression 3.14 where Q_k is the list of section classes that belong to the same grade as k, as defined in Expression 3.15.

$$\nu_S(q,k) = Card(\{(s,q) \in \mu : k \in SK(s)\}) \tag{3.13}$$
$$\overline{QN}(k) = Card(\{s \in S : k \in SK(s)\})/(Card(Q_k)) \tag{3.14}$$

Where

$$Q_k = \{q \in Q : KG(QK(q)) = KG(k) \wedge KT(QK(q)) = \text{“}section\text{”}\}) \tag{3.15}$$

Definition 3.5.3. Two section classes $q_i, q_j \in Q$ that belong to the same grade, i.e. $QK(q_i) = QK(q_j)$, are *similar for a category* k if they have the same number of students for this category, that is if $\nu_S(q_i,k) = \nu_S(q_j,k)$.

Definition 3.5.4. The section classes of a grade are *similar* if they are all similar for any category, that is if Expression 3.16 is satisfied.

$$\forall q_i, q_j \in Q, k \in K : QK(q_i) = QK(q_j) \Rightarrow \nu_S(q_i, k) = \nu_S(q_j, k) \tag{3.16}$$

Definition 3.5.5. The *absolute deviation* of a category $k \in K$ is defined in Expression 3.17 and measures how far this category is from the optimal similarity.

$$AbsDev(k) = \frac{1}{Card(GS(k))} \sum_{q \in GS(k)} |\nu_S(q, k) - \overline{QN}(k)| \tag{3.17}$$

The similarity objective is modeled with the minimization of the sum of the absolute deviation for all categories as in defined in Expression 3.18.

$$Cost(\mu) = \sum_{k \in K} AbsDev(k) \tag{3.18}$$

3.6 The CSOP model for the CLAP

For the CSOP model, the variables $X = \{X_1, \ldots, X_n\}$ are the section class allocated to students, where $n = Card(S)$ is the number of students, e.g. X_s is the section class allocated to student s. The domain D_s for each variable $X_s \in X$ is the list of the section classes corresponding to the student s as defined in Expression 3.19. The constraints of this model are the capacity of the classes: the number of students allocated to a class cannot exceed its capacity as defined in Expression 3.20.

$$\forall s \in S \quad : \quad D_s = \{q \in Q : QK(q) \in SK(s) \wedge KT(QK(q)) = \text{“}section\text{”}\} \tag{3.19}$$
$$\forall q \in Q \quad : \quad Card(\{X_i \in X : X_i = q\}) \leq KS_{max}(QK(q)) \tag{3.20}$$

3.7 The ACO approach for the CLAP

In the ACO algorithm 6 presented in this chapter, each assignment (s, q) of a student to a section class has a quantity of pheromones on it which is used to calculate the probability of including this assignment in the solution of an ant. At each iteration, each ant of the colony builds a solution step by step selecting one assignment (s, q) for each student, lines 10 to 13. At the beginning of the algorithm, all pheromones $\tau_{s,q}$ are initialized to a given value, line 4. At the end of an iteration, part of the pheromones evaporate, which means that the value of $\tau_{s,q}$ decreases, line 18. The fitness of the solutions found by the ants during the iteration is computed and compared, lines 14 to 16. The best ant deposits pheromones on the assignments (s, q) that belong to its solution, the value of $\tau_{s,q}$ increases, line 18.

Algorithm 6: Model. The ACO algorithm

Input: List of students with their categories, List of section classes with their capacity
Output: Allocation μ: Assignment of each student to one section class

$\mu \leftarrow \emptyset$ **for** $q \in Q$ **do**
  $\eta_q = 1$ `// Visibility initialisation`
for $(s, q) \in (S \times Q)$ **do**
  $\tau_{s,q} = \tau_{init}$ `// Pheromones initialisation`
for $i = 1, \ldots, NbIterations$ **do**
  $\mu_{iter} \leftarrow \emptyset$
  for $j = 1, \ldots, NbAnts$ **do**
    $\mu_{ant} \leftarrow \emptyset$
    for $s = 1, \ldots, NbStudents$ **do**
      Select a section class q of the grade of s
      $\mu_{ant} \leftarrow \mu_{ant} \cup (s, q)$
      if $Card(\{(s_i, q) \in \mu_j\}) = KS_{max}(QK(q))$ **then**
        $\eta_q = 0$ `// Visibility update`
    Calculate the fitness function $f(\mu_{ant}) = \sum_{k \in K} AbsDev(k)$
    if $f(\mu_{ant}) < f(\mu_{iter})$ **then**
      $\mu_{iter} \leftarrow \mu_{ant}$
  for $(s, q) \in (S \times Q)$ **do**
    $\tau_{s,q} \leftarrow (1 - \rho) \cdot \tau_{s,q}$ `// Pheromones evaporation`
  for $(s, q) \in \mu_{iter}$ **do**
    $\tau_{s,q} \leftarrow \tau_{s,q} + \frac{\varphi}{f(\mu_{iter})}$ `// Pheromones deposition`
  if $f(\mu_{iter}) < f(\mu)$ **then**
    $\mu \leftarrow \mu_{iter}$

In addition to the pheromones, the ants also use the visibility η_q they have on their own partial solution. Indeed when it is building a solution, each ant knows which assignments are in its partial solution and the capacity of the classes. As classes should never be overbooked, the ant has to know somehow if a class can still be assigned to a student or not. The visibility is updated after each allocation of a student to a class and its value is given by Expression 3.21. The probability to select a section class q for a student s is given by Expression 3.22, where Q_s is the list of section classes that belong to the same grade as s, as defined in Expression 3.23.

$$\eta_q = \begin{cases} 1 & \text{if there are still seats available in } q \\ 0 & \text{if } q \text{ has no more seats available} \end{cases} \tag{3.21}$$

$$p(s, q) = \frac{\tau_{s,q} \cdot \eta_q}{\sum_{i \in Q_s} \tau_{s,i} \cdot \eta_i} \tag{3.22}$$

$$Q_s = \{q \in Q : QK(q) \in SK(s) \wedge KT(QK(q)) = \text{“}section\text{”}\} \tag{3.23}$$

Grade		School 2112	1213	3411	3621
9-VG	Students	113	71	78	68
	Section classes	6	4	5	4
	Categories	16	14	15	17
10-VG	Students	137	90	78	66
	Section classes	8	5	5	4
	Categories	15	16	15	16

Table 3.3: Size of the datasets used.

When an assignment (s, q) belongs to the best solution of an iteration, pheromone is deposited on it at the end of the iteration, the values of $\tau_{s,q}$ and therefore of $p(s, q)$ are increased. In the next iteration, q has a higher probability to be allocated to s than in the iteration that just finished. The parameters used in this ACO algorithm are the number of iterations, the number of ants, the evaporation rate ρ, and the deposition rate φ.

3.8 Experimentation

The eight datasets used in this chapter come from the Direction Générale de l'Enseignement Obligatoire from Canton de Vaud in Switzerland. They correspond to two different grades of four schools and to the academic period 2015-2016. The available data is the list of students with the categories they belong to and the number of section classes per grade. One student belongs usually to 5 categories, few of them belong to only 3 or 4. Table 3.3 contains the number of students, of section classes and of categories per school and per grade-section.

The computation was done on an Intel® Core™i5-5300U CPU 2.30 GHz. Gecode was used to solve the CSOP model, [103]. Gecode has been configured to perform parallelized simulations with three threads while the ACO approach runs simulations in series on a single processor. In Gecode, the selection of the variables is always the same and depends only on the sequence of students in the file, the selection of their value for the branching is done randomly, and the runtime is 10 minutes. 80 iterations with 60 ants are used for the ACO approach, one run needs between 1 and 3 seconds.

The algorithms were launched 10 times, with different seeds each time, for each dataset in order to have average and standard deviation. Table 3.4 contains the results: average, standard deviation, best and worst results. Regarding the average, ACO clearly outperforms the CSOP solver, which is between 14% and 44% worse. Regarding the best solution found among the 10 runs, the CSOP solver outperforms or is identical to ACO in four datasets.

Figure 3.2 shows how students are spread in the different section classes for each category in a solution found by ACO. For each category, the closer the points are, the better the solution is. Category 1 corresponds to the section category. For category 1 in school 3621, the size of the section classes is between 18 and 19 students in Figure 3.2, and they are thus similar. For the other categories, the difference in the number of students per class is between 1 and 4, e.g. in category 7, class 3 has 9 students of this category and class 2 has 12 of them.

		ACO			CSOP 10 minutes		
School	**Grade**	**Av.±St. Dev.**	**Best**	**Worst**	**Av.±St. Dev.**	**Best**	**Worst**
2112	9-VG	58.4 ± 5.7	52	68	79.0 ± 10.2	62	96
	10-VG	75.2 ± 4.8	67	81	103.0 ± 19.0	61	139
1213	9-VG	26.8 ± 4.1	22	34	37.2 ± 5.7	30	46
	10-VG	39.2 ± 5.3	33	47	56.6 ± 13.5	33	79
3411	9-VG	38.2 ± 3.8	34	44	51.4 ± 4.3	44	58
	10-VG	37.8 ± 3.5	33	45	51.4 ± 9.9	37	69
3621	9-VG	29.2 ± 3.5	22	34	33.2 ± 8.2	22	46
	10-VG	27.0 ± 2.9	22	32	36.6 ± 8.9	22	56

Table 3.4: Results: average, standard deviation, best and worst result with 10 runs.

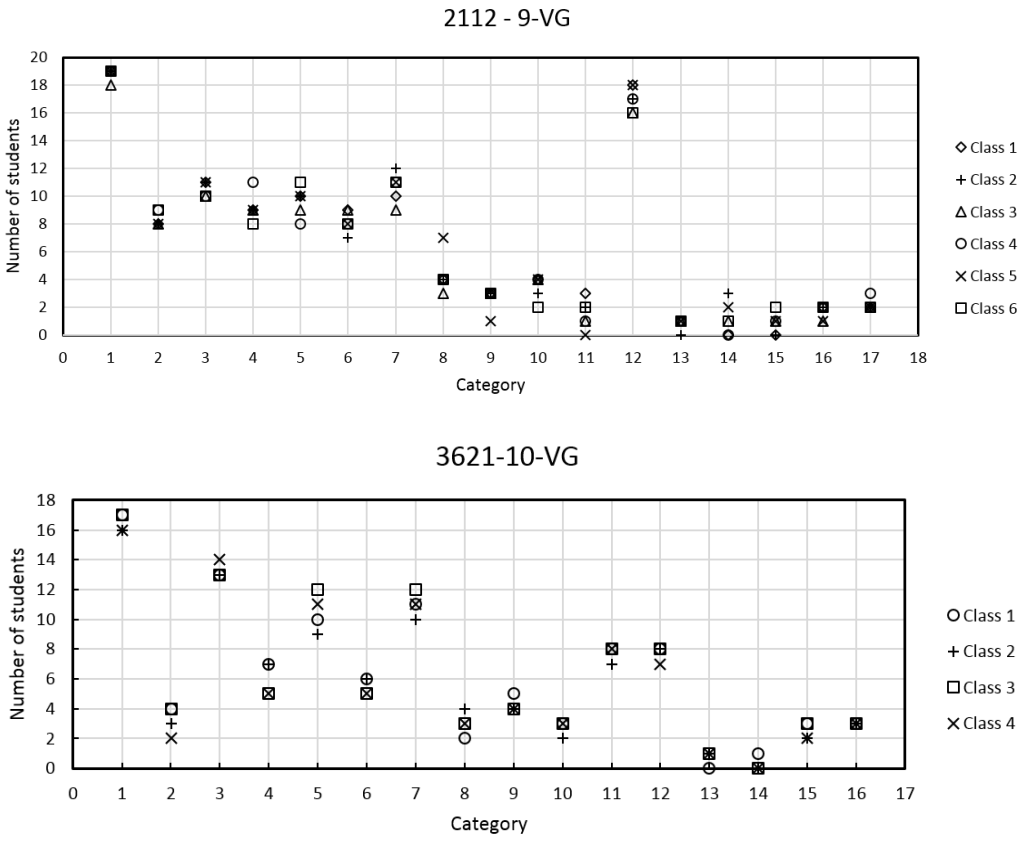

Figure 3.2: Results for schools 2112-9-VG and 3621-10-VG, with ACO.

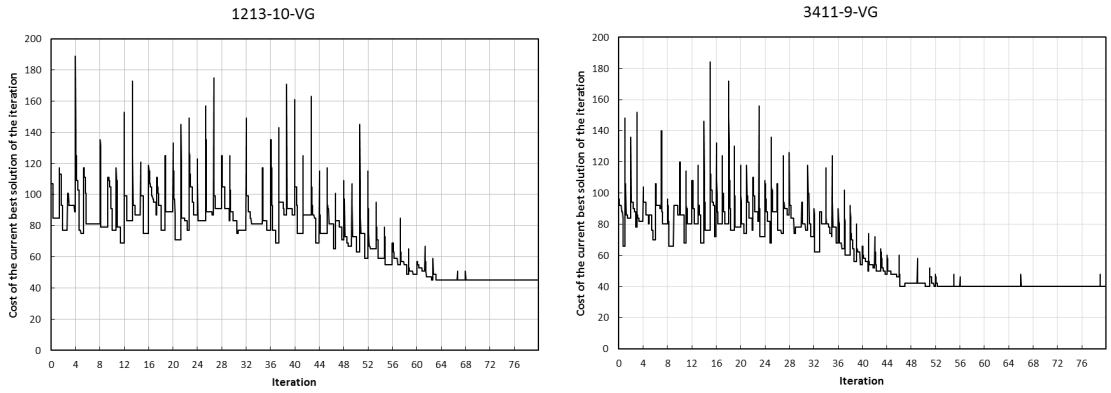

Figure 3.3: Evolution of the solution cost for the ACO algorithm in 1 run.

Figure 3.3 shows the cost of the current best solution during one run of the ACO algorithm for two different sections of two schools. The convergence takes place after 1 to 3 seconds, depending on the school and the grade considered.

3.9 Conclusions

In this chapter we have developed a constraint-based model for the CTT problem in secondary schools that have two types of courses: section courses that are defined by a curriculum and are taught to a whole class, and courses that are not taught per class, but to groups of students who have the same level in a specific subject or have selected the same elective course. Besides that, the model includes the fact that students are allocated to classes at the same time as the timetable is built. Both problems, allocation and timetabling are linked since the way students are allocated to classes has a direct impact on the timetable constraints.

We have also presented two approaches to solve the class allocation problem modeled as a resource allocation problem. The objective is to have all classes with a similar composition for all topics, which means that students with the same profile on a topic have to be spread among the available classes. This maximizes also the diversity of each class, since a maximum number of different profiles are allocated to each of them.

Two approaches have been applied to eight real datasets coming from schools located in Canton de Vaud. They are based on ACO and on CSOP. The ACO algorithm provides better solutions than the CSOP solver in a shorter time. As both of them use randomness for the allocation of resources, our results prove that the pheromones in the ACO approach help to find very good solutions in a much smaller amount of time.

As the computation time is short for the ACO approach, the school's directors, who are in charge of the allocation of students and of the timetable, can simulate multiple scenarios of possible sets of profiles in their school. They can then analyze the impact of those scenarios on the pedagogical objective with the solution proposed in this chapter and on the timetable problem with the commercial software available in the different schools.

3.10 Case Study: The Course Timetabling Problem

This section describes one variant of the CTT problem mentioned in Section 2.3.1.2 and the method used to tackle it, two ACO algorithms combined with a local search. This metaheuristic will then be applied to the datasets of the International Timetabling Competition ITC-2007 so that its performance can be compared to the different published results, [111]. Table 3.5 contains a sum-up of the notation used in this chapter.

Notation	Description
$q \in Q$	Curricula of the CTT problem.
$c \in C$	Courses of the CTT problem.
$Q(c) \subset Q$	Curricula to which the course $c \in C$ belongs, $Q(c) \neq \emptyset$, i.e. c belongs to at least one curriculum.
$N(c)$	Number of students in course $c \in C$.
$T(c) \in T$	Teacher of course $c \in C$
$D(c)$	Minimum number of days which at least one lecture of $c \in C$ is allocated to.
$p \in P$	Periods of the CTT problem, periods are numbered from 1 to $Card(P)$.
$D(p)$	Day of the period $p \in P$, if $p_1 < p_2$, then $D(p_1) \leq D(p_2)$.
$l \in L$	Lessons of the CTT problem.
$C(l) \in C$	Course of lesson $l \in L$; $\forall l \in L, \exists c \in C : c(l) = c$.
$t \in T$	Teachers of the CTT problem.
$TP(t) \subset P$	List of periods where $t \in T$ is available.
$r \in R$	Rooms of the CTT problem.
$CAP(r)$	Capacity of room $r \in R$.

Table 3.5: Notation for the CTT problem.

3.10.1 The ITC-2007 Course Timetabling Problem

In the ITC-2007 instances, the timetable is built for a typical week. This week is structured in five or six days and each day contains several periods. Figure 3.4 contains an example of a week structure with five days and 20 periods spread among the five days.

Timeslot	Day 1	Day 2	Day 3	Day 4	Day 5
1	Period 1	Period 5	Period 9	Period 13	Period 17
2	Period 2	Period 6	Period 10	Period 14	Period 18
3	Period 3	Period 7	Period 11	Period 15	Period 19
4	Period 4	Period 8	Period 12	Period 16	Period 20

Figure 3.4: Example of a week structure: 5 days with 4 periods per day.

A course $c \in C$ has a specific number of students $N(c)$ and one teacher $T(c) \in T$ assigned. A course has n_c lectures $\{l_c^1, \ldots, l_c^{n_c}\} \subset L$ and each lecture $l \in L$ corresponds to a course $c \in C$. A course belongs to one or several curricula and a curriculum is made of one or several courses. Each lecture has to be allocated to a room $r \in R$ and to a period $p \in P$, i.e. a day $d \in D$ and a timeslot $s \in S$.

Definition 3.10.1. A solution to a CTT problem is made of two allocations μ and ν such that $\forall l \in L$:

$\mu(l) \in R$ Each lesson is assigned to a room.
$\nu(l) \in P$ Each lesson is assigned to a period.

When a timetable is built, two types of constraints have to be considered: hard and soft constraints. All the constraints mentioned below come from the rules set for ITC-2007.

The following hard constraints must be satisfied in order to have a complete solution:

- All lectures of a course must be scheduled, and they must be assigned to distinct periods:

$$\forall l_1, l_2 \in L : \qquad C(l_1) = C(l_2) \Rightarrow \mu(l_1) \neq \mu(l_2) \tag{3.24}$$

- Two lectures cannot take place in the same room in the same period:

$$\forall l_1, l_2 \in L : \qquad \nu(l_1) = \nu(l_2) \Rightarrow \mu(l_1) \neq \mu(l_2) \tag{3.25}$$

- Lectures of courses in the same curriculum or taught by the same teacher must be all scheduled in different periods:

$$\forall l_1, l_2 \in L : \qquad Q(C(l_1)) = Q(C(l_2)) \Rightarrow \mu(l_1) \neq \mu(l_2) \tag{3.26}$$

$$\forall l_1, l_2 \in L : \qquad T(C(l_1)) = T(C(l_2)) \Rightarrow \mu(l_1) \neq \mu(l_2) \tag{3.27}$$

- The lectures of a course have to be allocated to periods when the teacher of the course is available:

$$\forall l \in L : \qquad \nu(l) \in TP(T(C(l))) \tag{3.28}$$

Definition 3.10.2. For a solution (μ, ν), $Hard(\mu, \nu)$ is the number of hard constraints that are not satisfied. A solution to a CTT problem is *feasible* if all constraints 3.24 to 3.28 are satisfied, i.e. $Hard(\mu, \nu) = 0$.

A maximum number of the following soft constraints should be satisfied. For each constraint that is not satisfied, a penalty is applied. A solution has therefore a cost associated to it, the sum of all the penalties related to the violation of the soft constraints. The value of those penalties were set for the ITC-2007.

- For each lecture, the number of students that attend the course should be less or equal than the number of seats of all the rooms that host its lectures

$$\forall l \in L : \qquad \nu(l) = r \Rightarrow N(C(l)) \leq CAP(r) \tag{3.29}$$

Penalty: Each student above the capacity counts as 1 point. The total amount of those penalty points correspond to the cost C_{CR}

- The lectures of each course should be spread into a minimum number of days

$$\forall c \in C : \qquad Card(\{d \in D : \exists l \in L : C(l) = c \wedge D(\mu(l)) = d\}) \geq D(c) \tag{3.30}$$

Penalty: Each day below the minimum counts as 5 points. The total amount of those penalty points correspond to the cost C_{MD}

- Lectures belonging to a curriculum should be adjacent to each other (i.e., in consecutive periods) if those lectures are assigned to the same day

$$\forall d \in D, q \in Q : \qquad Card(\lambda_{QD}(q,d)) > 1 \Rightarrow \forall l_1 \in \lambda_{QD}(q,d) : \exists l_2 \in \lambda_{QD}(q,d) : \mu(l_1) = \mu(l_2) \pm 1 \tag{3.31}$$

where $\lambda_{QD}(q,d)$ is the set of lessons that belong to a course of curriculum q and are allocated to day d:

$$\lambda_{QD}(q,d) = \{l \in L : Q(C(l)) = q \wedge D(\mu(l)) = d\} \tag{3.32}$$

and $Card(\cdot)$ denotes cardinality of a set. **Penalty**: Each isolated lecture in a curriculum counts as 2 points. The total amount of those penalty points correspond to the cost C_{IL}

- All lectures of a course should be given in the same room

$$\forall l_1, l_2 \in L : \qquad C(l_1) = C(l_2) \Rightarrow \nu(l_1) = \nu(l_2) \tag{3.33}$$

Penalty: Each distinct room used for the lectures of a course, but the first, counts as 1 point. The total amount of those penalty points correspond to the cost C_{NR}

Definition 3.10.3. A feasible solution to a CTT problem, μ and ν has an associated cost given by:

$$Cost(\mu,\nu) = C_{CR} + C_{MD} + C_{IL} + C_{NR} \tag{3.34}$$

Considering definitions 3.10.2 and 3.10.3, a solution (μ_1, ν_1) is better than a solution (μ_2, ν_2) if one of those situations is true:

- $Hard(\mu_1, \nu_1) < Hard(\mu_2, \nu_2)$
- $Hard(\mu_1, \nu_1) = Hard(\mu_2, \nu_2)$ and $Cost(\mu_1, \nu_1) < Cost(\mu_2, \nu_2)$

3.10.2 The ACO approach

In the approach presented in this chapter, two colonies of ants assign rooms and periods to the lectures. One colony is dedicated to the room's assignment and the other to the period's assignment. Each colony has its own pheromones.

Algorithm 7 describes the initialization of the pheromones and the visibility. All pheromones are initialized to an initial value τ_{init}. Regarding rooms allocation, the visibility for a lecture is smaller for rooms with insufficient capacity than for the other rooms. The visibility of a period

Lecture	Day 1	Day 2	Day 3	Day 4	Day 5
1					
2					
3					

Figure 3.5: Example of visibility difference when $D(c) > 1$. Periods belonging to the white days have a smaller visibility for the corresponding lecture.

for a lecture is smaller if the corresponding teacher is not available on this period. The periods visibility also considers the soft constraint 3.30 which requires a minimum number of days where the lectures of a course are spread. If there is such a minimum, i.e. $D(c) > 1$, then $D(c)$ lectures of the course are selected randomly and each of them has a smaller visibility on $D(c) - 1$ days. Figure 3.5 contains an example with 5 days in the week and $D(c) = 3$, then 3 lectures of c are selected randomly and their visibility is reduced on 2 different days for each of them.

Algorithm 8 describes the ACO algorithm. At every iteration, each ant of the colony builds a solution (μ, ν). This solution is then locally improved, line 12, which consists in trying to swap the lectures that break a hard constraint with the rest of the lectures. Two lectures are swapped by exchanging their room and their period. If the swapped solution is better than the initial solution, the swapped one is kept and the swapping continues until all hard-breaker lectures are tested for the improvement.

A the end of the iteration, the best solution of the iteration is also locally improved, line 17. This local search is slightly different from the ant improvement since all lectures are candidates for a swap. This swap is followed by a room-swap where the period is not changed, only the room exchange is considered, so the two candidates of a swap have to be assigned to the same period.

3.10.3 The datasets

Table 3.6 contains the size of the 21 datasets that were used for the ITC-2007: number of rooms, of periods, of courses, of lectures and of curricula. The number of constraints indicated in the last column corresponds to the total number of periods when a teacher is not available, that is when a lecture cannot be assigned.

3.10.4 The results

The ACO approach was launched 10 times with 500 iterations each and a colony of 30 ants. The results are presented in table 3.7.

All solutions found are feasible, except for the dataset 05, where only four runs produced feasible solutions, five runs produced a solution with one broken hard constraint and one run produced a solution with two broken hard constraints. In dataset 11, the ACO approach produced several solutions with a cost 0, that is as good as the best known solution.

The bad quality of the results come mainly from the local search that does not consider

Algorithm 7: Pheromones and visibility initialization

Input: Dataset of the ITC-2007

Output: Allocations μ and ν: Assignment of each lecture to one room and one period

for $(l, r) \in (L \times R)$ **do**
  $\tau_R(l, r) = \tau_{init}$ `// Rooms pheromones initialization`
  `// Rooms visibility initialization`
  if $CAP(r) < N(C(l))$ **then**
    $\eta_R(l, r) = CAP(r) * \nu_{min}$ `// Room with insufficient capacity`
  else
    $\eta_R(l, r) = N(C(l)) * \nu_{max}$ `// Room with sufficient capacity`
for $(l, p) \in (L \times P)$ **do**
  $\tau_{l,p}^{period} = \tau_{init}$ `// Periods pheromones initialization`
  `// Periods visibility initialization`
  if $p \in TP(T(C(l)))$ **then**
    $\eta_P(l, p) = 1$ `// Teacher available`
  else
    $\eta_P(l, p) = 0.1$ `// Teacher not available`
  $\tau_P(s, q) = \tau_{init}$ `// Pheromones initialization`
`// Visibility includes information about soft constraint 3.30`
for $c \in C$ **do**
  if $D(c) > 1$ **then**
    Select $D(c)$ lectures of $c \rightarrow L_d(c)$
    Reduce to 0.5 the visibility of $l \in L_d(c)$ on $D(c) - 1$ days
`// Calculate probabilities`
for $(l, r) \in (L \times R)$ **do**
  $$Prob_R(l, r) = \frac{(\tau_R(l, r))^\alpha \cdot (\eta_R(l, r))^\beta}{\sum_{r_i \in R} (\tau_R(l, r_i))^\alpha \cdot (\eta_R(l, r_i))^\beta}$$
for $(l, p) \in (L \times P)$ **do**
  $$Prob_P(l, p) = \frac{(\tau_P(l, p))^\alpha \cdot (\eta_P(l, p))^\beta}{\sum_{p_i \in R} (\tau_P(l, p_i))^\alpha \cdot (\eta_P(l, p_i))^\beta}$$

Algorithm 8: The ACO algorithm for the CTT problem

Input: Dataset of the ITC-2007

Output: Allocations μ and ν: Assignment of each lecture to one room and one period

Visibility initialization
for $i = 1, \ldots, NbIterations$ **do**
  $\mu_{iter} = \emptyset$
  for $j = 1, \ldots, NbAnts$ **do**
    $\mu = \emptyset$
    $\nu = \emptyset$
    for $l = 1, \ldots, Card(L)$ **do**
      Select a period p for l with probability $Prob_P(l, p)$
      $\mu = \mu \cup (l, p)$
      Select a room r for l with probability $Prob_R(l, r)$
      $\nu = \mu \cup (l, r)$
    Improve locally (μ, ν)
    Calculate $Hard(\mu, \nu)$ and $Cost(\mu, \nu)$
    `// The best solution of the iteration is updated if the solution found by the ant is better`
    if (μ, ν) *is better than* (μ_{iter}, ν_{iter}) **then**
      $\mu_{iter} = \mu$
      $\nu_{iter} = \nu$
  Improve locally (μ_{iter}, ν_{iter})
  `// The global best solution is updated if the solution of the iteration is better`
  if (μ_{iter}, ν_{iter}) *is better than* (μ_{best}, ν_{best}) **then**
    $\mu_{best} = \mu_{iter}$
    $\nu_{best} = \nu_{iter}$
    Improve locally (μ_{best}, ν_{best})
  else
    Improve locally (μ_{best}, ν_{best})
  for $(l, p) \in (L \times P)$ **do**
    $\tau_P(l, p) = (1 - \rho_P) \cdot \tau_P(l, p)$ `// Pheromones evaporation`
  for $(l, p) \in \mu_{iter}$ *and* $(l, p) \in \mu_{best}$ **do**
    $\tau_P(l, p) = \tau_P(l, p) + \frac{\varphi_P}{Cost(\mu_{iter}, \nu_{iter})}$ `// Pheromones deposition`
  for $(l, r) \in (L \times R)$ **do**
    $\tau_R(l, r) = (1 - \rho_R) \cdot \tau_R(l, r)$ `// Pheromones evaporation`
  for $(l, r) \in \nu_{iter}$ *and* $(l, r) \in \nu_{best}$ **do**
    $\tau_R(l, r) = \tau_R(l, r) + \frac{\varphi_R}{Cost(\mu_{iter}, \nu_{iter})}$ `// Pheromones deposition`
  Calculate the probabilities $Prob_P(l, p)$ and $Prob_R(l, r)$ as in algorithm 7

Dataset	Rooms	Days	Per./Day	Periods	Curr.	Courses	Lectures	Constraints
01	6	5	6	30	14	30	160	53
02	16	5	5	25	70	82	283	513
03	16	5	5	25	68	72	251	382
04	18	5	5	25	57	79	286	396
05	9	6	6	36	139	54	152	771
06	18	5	5	25	70	108	361	632
07	20	5	5	25	77	131	373	667
08	18	5	5	25	61	86	324	478
09	18	5	5	25	61	86	324	478
10	18	5	5	25	67	115	346	694
11	5	5	9	45	13	30	162	94
12	11	6	6	36	150	88	218	1368
13	19	5	5	25	66	82	308	468
14	17	5	5	25	60	85	275	486
15	16	5	5	25	68	72	251	382
16	20	5	5	25	71	108	366	518
17	17	5	5	25	70	99	339	548
18	9	6	6	36	52	47	138	594
19	16	5	5	25	66	74	277	475
20	19	5	5	25	78	121	375	691
21	18	5	5	25	78	94	327	463

Table 3.6: Size of the datasets of the ITC-2007.

how the costs are computed. For example, the cost C_{NR} that corresponds to the number of rooms used for a course is seldom improved if lectures are considered individually as in the swap process. When half of the lectures are assigned to one room and the other half to another room, then moving one single lecture does not improve that cost; all lectures assigned to one room should be simultaneously moved to the other room.

Figure 3.6 shows the cost evolution for the best solution of the 500 iterations for one of the datasets. A clear convergence is visible, even if the result is three times worse from the best solution found in the literature.

3.10.5 Conclusions

The first results obtained with the ACO approach and a simple local search are encouraging, but far from specific and optimized algorithms developed especially for the ITC-2007 datasets. Feasible solutions have always been found, that means with no hard constraint broken.

To improve quality and performance of the ACO approach, two aspects must be improved. First the local search must be adapted to consider how the costs are computed. Second the parameters of ACO have to be tuned to speed up the computation and the convergence.

Dataset	Results (Av. ± Std Dev.)	Best	Best known
01	9 ± 2.3	6	5
02	147 ± 33.1	56	10
03	158 ± 17.8	127	38
04	103 ± 9.5	88	35
05*	640 ± 72.5*	487*	114
06	138 ± 13.5	109	16
07	110 ± 9.8	92	6
08	106 ± 12.6	89	37
09	108 ± 11.7	89	66
10	110 ± 14.0	91	4
11	3 ± 1.7	0	0
12	548 ± 60.2	492	53
13	132 ± 9.1	115	48
14	141 ± 8.9	126	51
15	158 ± 17.8	127	41
16	134 ± 11.2	120	13
17	159 ± 8.5	151	44
18	171 ± 20.6	143	0
19	193 ± 13.0	169	49
20	162 ± 13.9	138	0
21	195 ± 15.7	171	0

Table 3.7: Results in the different datasets of the ITC-2007. All solutions are feasible, except dataset 05.

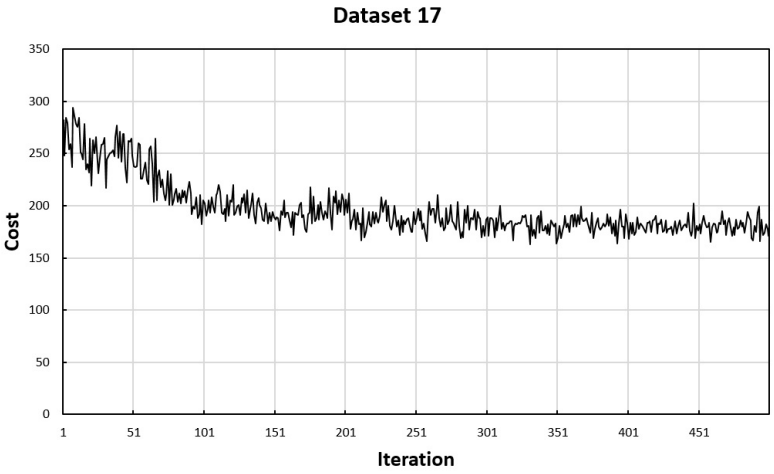

Figure 3.6: Evolution of the best solution cost for the dataset 17.

Chapter 4

Heuristic and Hybrid Metaheuristic Approaches for RAP Combined with a Scheduling Problem

This chapter describes a Course Allocation Problem combined with a Course Scheduling Problem found at the Ecole Hôtelière de Lausanne. Both problems, the Course Allocation and the Course Scheduling, are first formally defined with their constraints and their objective. Two approaches are then proposed to solve the Course Allocation problem, a greedy algorithm and a CSOP approach. Both of them are compared using the real data from the Ecole Hôtelière de Lausanne. The latter is then adapted to solve the combined problem and compared to an ACO approach using the same data and adding the constraints related to the Course Scheduling Problem.

4.1 Introduction

The **course allocation problem** (CAP) is critical in many universities and business schools. When students can select the courses they would like to attend, they often submit preferences, but have no guarantee that they will get a seat in all the courses in which they want to enroll due mainly to quality and security constraints, that limit the number of seats available in a course, [22].

In the scheduling problem presented in this chapter, there is no curriculum, students can attend any combination of courses, but they submit preferences, they do not select courses, [106].

Two methods are proposed to solve the CAP: the first is based on a greedy approach, the second is based on CSOP. The generation of our sample data is based on courses offered to students in their final semester at the Ecole Hôtelière de Lausanne (EHL); students indicate their preferences with a ranking of all the available courses.

- The **Course Greedy Algorithm** (CGA) assigns students to courses so that a course is filled with students who ranked it in their first choices.
- The **CSOP** approach assigns students to courses so that the global welfare of students is maximized and the satisfaction of the worst off is optimized.

Two models CAP&CTT are proposed and include both problems, the allocation of students to courses considering their preferences and the timetabling of those courses. Both problems, CAP and CTT, are solved simultaneously so that the allocation of students to courses is compatible with a feasible timetable.

The first approach proposed is based on CSOP. Both the CAP and the CTT are modeled as CSOP and solved with the solver Gecode, [131]. The objective is to find a solution that optimizes the students' satisfaction and that can be scheduled considering the timetabling constraints.

In the second approach an ACO algorithm is used to solve the timetabling problem, [43, 103, 135]. Each ant is going to look for a feasible timetable, the timetable becomes thus a constraint for the CAP, which is then solved with Gecode. The objective for the ant colony is to find a timetable that allows the CSOP solver to provide good solutions to the CAP.

The rest of the chapter is structured as follows. Section 4.2 describes the CAP and the metrics used to compare different allocations. Section 4.3 is dedicated to the CTT. Sections 4.4 and 4.5 describe the CGA and the CSOP model for the CAP. Sections 4.6 and 4.7 contain the CSOP model and the ACO approach used to solve the CAP&CTT. Section 4.8 presents the results of the two approaches CGA and CSOP applied to solve the CAP. Section 4.9 provides the results of the two approaches CSOP and ACO for the combined CAP&CTT for three real datasets from EHL. Finally in Section 4.10 the conclusions are exposed.

In this chapter, the following notation is used:

- $S = \{s_1, \ldots, s_n\}$ is the list of students and $G(s)$ is the average grade of student s;
- $C = \{c_1, \ldots, c_m\}$ is the list of available courses and $K(c)$ is the capacity of course c, i.e. a maximum of $K(c)$ students can be assigned to course c;
- γ is the number of courses that should be assigned to each student. Enough seats are available to allocate γ courses to each of the n students, that means

$$\sum_{c \in C} (K(c)) \geq n \cdot \gamma \tag{4.1}$$

4.2 The Course Allocation Problem (CAP)

In most business schools, elective courses with a limited number of seats are allocated to students. Each student has to rank the available courses and is assigned to courses depending on his preferences. Without loss of generality, we may consider that the number of courses allocated is the same for all students. A formal definition of the problem is presented in this section.

Definition 4.2.1. The *ranking* of a student $s \in S$ is a relation $\succeq_s$, such that $\forall c_i, c_j \in C$:

$$\begin{aligned} c_i \succ_s c_j \quad &\Leftrightarrow \quad s \text{ strictly prefers } c_i \text{ over } c_j \\ c_i =_s c_j \quad &\Leftrightarrow \quad s \text{ has no preference of } c_i \text{ over } c_j \text{ or } c_j \text{ over } c_i \end{aligned}$$

A ranking is said to be total if all courses are ranked by all students.

Definition 4.2.2. The *rank* of a student $s \in S$ for a course $c \in C$ is a function $Rank_s$, such that $\forall c, c_i, c_j \in C, \forall s \in S$:

$$\begin{aligned} &Rank_s(c_i) < Rank_s(c_j) \quad &\Leftrightarrow \quad c_i \succ_s c_j \\ &Rank_s(c_i) = Rank_s(c_j) \quad &\Leftrightarrow \quad c_i =_s c_j \\ &Rank_s(c) \in \{0, 1, ..., Card(C)\} \end{aligned}$$

Definition 4.2.3. An *allocation* μ of courses to students is a function such that $\forall c \in C, \forall s \in S$:

$$\begin{aligned} &\mu(c) = \{s^c_1, \ldots, s^c_{Card(\mu(c))}\} \quad &\text{is the list of students assigned to } c \\ &\mu^{-1}(s) = \{c^s_1, \ldots, c^s_{Card(\mu^{-1}(s))}\} \quad &\text{is the list of courses allocated to } s \end{aligned}$$

Definition 4.2.4. An allocation μ of courses to students is said to be *feasible* if

$$\begin{aligned} &\forall c \in C : \quad &Card(\mu(c)) \leq K(c) \\ &\forall s \in S : \quad &Card(\mu^{-1}(s)) = \gamma \\ &\forall s_1, s_2 \in \mu(c) : \quad &s_1 \neq s_2 \\ &\forall c_1, c_2 \in \mu^{-1}(s) : \quad &c_1 \neq c_2 \end{aligned}$$

Definition 4.2.5. The *position* $P_s(c)$ of a course $c \in C$ for a student $s \in S$ is defined by the number of courses that s strictly prefers over c:

$$P_s(c) = Card(c_i \in C : c_i \succ_s c) \tag{4.2}$$

The favorite courses of a student have a position 0, since there are no courses preferred over them.

Definition 4.2.6. A student s has a *strict ranking* if

$$\forall c_i, c_j \in C \text{ with } i \neq j : c^s_i \prec_s c^s_j \text{ or } c^s_i \succ_s c^s_j \text{ which means } P_s(c_i) \neq P_s(c_j) \tag{4.3}$$

With a strict ranking, there is no indifference: for any pair of courses, a student always strictly prefers one of them over the other, [124].

Definition 4.2.7. For an allocation of courses μ, the *satisfaction gap* $SatGap_\mu(s)$ of a student $s \in S$ is defined by

$$SatGap_\mu(s) = \sum_{c \in \mu^{-1}(s)} P_s(c) \tag{4.4}$$

For each course allocated to a student, the satisfaction gap $SatGap_\mu(s)$ counts the number of courses that the student strictly prefers over this course. For example, let's consider a student

who is assigned to 2 courses whose positions are 5 and 3. That means that there are 5 courses that he prefers over the first course and 3 courses that he prefers over the second course. His satisfaction gap with this allocation is then 8. So the smaller $SatGap_\mu(s)$ is, the happier the student is.

Definition 4.2.8. For an allocation of courses μ, the *rank gap* $RankGap_\mu(s)$ of a student $s \in S$ is defined by

$$RankGap_\mu(s) = \sum_{c \in \mu^{-1}(s)} Rank_s(c) \tag{4.5}$$

So the smaller $Rank_\mu(s)$ is, the happier the student is.

The quality of an allocation can be measured with different parameters. The metrics used in this chapter to analyze the quality of an allocation μ are:

- **Total Satisfaction Gap (TSG)**. Sum of the satisfaction gap of all students.

$$(TSG) = \sum_{s \in S} SatGap_\mu(s) \tag{4.6}$$

  If the ranking is strict, then (TSG) has a lower bound: (TSG) $\geq n \cdot \frac{\gamma \cdot (\gamma - 1)}{2}$

- **Worst Satisfaction Gap (WSG)**. Worst satisfaction among students. The students whose satisfaction gap is equal to (WSG) are the worst off.

$$(WSG) = \max_{s \in S} SatGap_\mu(s) \tag{4.7}$$

- **Total Rank Gap (TRG)**. Sum of the rank gaps of all students.

$$(TRG) = \sum_{s \in S} RankGap_\mu(s) \tag{4.8}$$

- **Worst Rank Gap (WRG)**. The students whose rank gap is equal to (WRG) are the worst off.

$$(WRG) = \max_{s \in S} RankGap_\mu(s) \tag{4.9}$$

As the satisfaction of a student is the number of courses he prefers over the ones allocated to him, the objective is to minimize those metrics.

4.3 The Course Timetabling Problem (CTT)

In the CTT problem presented in this chapter, the resources are timeslots and teachers and the activities are the courses. Each course has a pre-assigned teacher. A formal definition of a timetable can be found in Definition 4.3.1. The compatibility of a timetable with an allocation is defined in Definition 4.3.2.

Definition 4.3.1. If TS is the list of the available timeslots, a *timetable* ν for the courses is a function such that

$\forall c \in C : \nu(c) \in TS$ is the time slot assigned to c

$\forall ts \in TS : \nu^{-1}(ts) = \{c^s_1, \ldots, c^s_{Card(\nu^{-1}(ts))}\}$ is the list of courses assigned to the timeslot ts

Definition 4.3.2. A timetable ν for the courses is *compatible* with an allocation μ if it contains no clash for students or teachers.

$\forall s \in S, \forall c_i, c_j \in \mu^{-1}(s) : c_i \neq c_j \Rightarrow \nu(c_i) \neq \nu(c_j)$

$\forall c_i, c_j \in C : Teacher(c_i) = Teacher(c_j) \wedge c_i \neq c_j \Rightarrow \nu(c_i) \neq \nu(c_j)$

4.4 The Course Greedy Algorithm (CGA) for the CAP

In this section, the number of available seats is identical to the number of needed seats, that is $\sum_{c \in C}(K(c)) = n \cdot \gamma$. To apply the CGA, students must be ordered using a specific criteria, in the following sections, students are ordered by their grades, ties are randomly broken.

In the first step of the CGA, the available seats of each course are allocated to the students who ranked the course first. In step k, only the courses that still have available seats are considered and those seats are allocated to the students who ranked the course in position k. If a student is already assigned to γ courses, he won't be considered in the following steps. This algorithm is sketched in Algorithm 9. To avoid infeasibility, a course becomes mandatory if the number of remaining seats is equal to the number of students who have not been assigned to enough courses.

Algorithm 9: Course Greedy Algorithm

Input: List of students $S = \{s_1, \ldots, s_n\}$ ordered by their grades:
$\forall i, j \leq n : i > j \Rightarrow G(s_i) \leq G(s_j)$

Output: Feasible allocation μ

```
1  ∀c ∈ C: μ(c) ← ∅
   /* Loop per position                                              */
2  for k ← 0 to m − 1 do
       /* Loop per course                                           */
3      for j ← 1 to m do
4          if Card({s ∈ S : μ^{-1}(s) < γ}) = K(c_j) − Card(μ(c_j)) then
               /* The course becomes mandatory to avoid infeasibility */
5              μ(c_j) ← μ(c_j) ∪ {s ∈ S : μ^{-1}(s) < γ}
6          if Card(μ(c_j)) < K(c_j) then
               /* Loop per student                                  */
7              for i ← 1 to n do
8                  if P_{s_i}(c_j) = k and Card(μ^{-1}(s_i)) < γ then
9                      μ(c_j) ← μ(c_j) ∪ {s_i}
```

	s	$P_s(c_1)$	$P_s(c_2)$	$P_s(c_3)$	$P_s(c_4)$
Ranking	1	0	2	2	0
	2	0	3	1	2
	3	1	2	0	3
	4	0	2	0	3
Results		$\mu(c_1)$	$\mu(c_2)$	$\mu(c_3)$	$\mu(c_4)$
		$\{s_1, s_2\}$	$\{s_3, s_4\}$	$\{s_3, s_4\}$	$\{s_1, s_2\}$

Table 4.1: Example. Ranking: Position of each course for all students: For each student s $\in$ {1,2,3,4}, $P_s(c_i)$ is the number of courses that s prefers over c_i
Results: Allocation of courses to students and metrics value for CGA: $\mu(c_i)$ is the list of students assigned to course c_i.

4.4.1 Example

Consider a CAP with $m = 4, \gamma = 2$ and four students $S = \{s_1, ..., s_4\}$ with $G(s_1) > G(s_2) > G(s_3) > G(s_4)$. Their rankings are given in Table 4.1.For example $P_1(c_1) = 0$ means that this course is the favorite course of s_1 together with c_4 since $P_1(c_4) = 0$. As $P_1(c_2) = P_1(c_3) = 2$, s_1 has no preference among those courses, but he prefers c_1 and c_4 over c_2 or c_3.

This table contains also the solution obtained when applying the CGA. $\mu(c_i)$ is the list of students assigned to course c_i, for example $\mu(c_1) = \{s_1, s_2\}$ means that s_1 and s_2 are assigned to course c_1. The results are $(TSG) = 6$ and $(WSG) = 2$.

4.5 The CSOP approach for the CAP

As defined in Section 2.2, a CSOP can be described with three sets and one function (X, D, C, f). The variables $X = \{x_j^i : i = 1, \ldots, n, j = 1, \ldots, \gamma\}$ for the CAP are the courses allocated to each student, e.g. $X_1 = \{x_1^1, \ldots, x_\gamma^1\}$ are the courses allocated to student 1. The domain D_i is the same for all the variables, it is the list of the available courses $\forall i = 1, \ldots, n : D_i = \{c_1, \ldots, c_m\}$.

The constraints of the model are of two types:

- **Unicity**: A course cannot be allocated more than once to a student

$$\forall s \in S : alldifferent(X_s) \tag{4.10}$$

- **Capacity**: The number of students assigned to a course cannot exceed its capacity

$$\forall c \in C : Card(x_i^j \in X : x_i^j = c) \leq K(c) \tag{4.11}$$

The objective function has to take into account the metrics defined in Section 4.2 with the following priorities:

- Priority 1: Minimize (TSG). The total satisfaction gap must be as small as possible, so that the global satisfaction of students is maximized.
- Priority 2: Minimize (WSG). The least satisfied student must be as satisfied as possible.

4.6 The CSOP approach for the CAP&CTT

In this section, the number of available seats is greater than or equal to the number of needed seats, that is $\sum_{c \in C}(K(c)) \geq n \cdot \gamma$. Therefore, courses with a low demand can be canceled.

As in Section 4.5, for the CAP, the variables X are the courses allocated to students and the domains D contain the list of the available courses.

There are three types of constraints, Unicity and Capacity as in Section 4.5, and a third constraint:

- **Compatibility**: Students cannot be allocated to courses that are not compatible

$$\forall \{c_1, \ldots, c_j\} \subset C \text{ not compatible}, \forall s \in S : \{c_1, \ldots, c_j\} \nsubseteq X_s$$

The objective function has to take into account the metrics defined in Section 4.2 with the following priorities:

Priority 1 Minimize (WRG). The least satisfied student must be as satisfied as possible.

Priority 2 Minimize (TRG). The total rank gap must be as small as possible, so that the global satisfaction of students is maximized.

Let us consider μ as an assignment of the CAP variables X defined previously. In order to apply these priorities, the objective function f_{CAP} is the linear function defined in Expression 4.12, where the weight $\alpha > 0$ used for (WRG) is higher than the weight $\beta > 0$ used for (TRG). This function f_{CAP} is defined in Expressions 4.12 and 4.13 and has to be minimized.

$$\begin{array}{rcll} f_{CAP} & = & \alpha \cdot (WRG) & + \quad \beta \cdot (TRG) \\ & = & \alpha \cdot \max_{s \in S} \left(RankGap_\mu(s)\right) & + \quad \beta \cdot \sum_{s \in S} \left(RankGap_\mu(s)\right) \end{array} \tag{4.12}$$

where

$$RankGap_\mu(s) = \sum_{i=1}^{\gamma} (Rank_s(x_i^s)) \tag{4.13}$$

For the CTT, the variables $Y = \{y_1, \ldots, y_m\}$ are the timeslots allocated to courses. The domain of y_i is the list of the possible slots. The constraints for the CTT are:

- **No-clashes for students**: If a student is allocated to courses c_i and c_j, those courses cannot be scheduled on the same slot.

$$\forall s \in S, \forall c_i, c_j \in X_s : y_{c_i} \neq y_{c_j}$$

- **No-clashes for teachers**: If c_i and c_j have the same teacher, those courses cannot be scheduled on the same slot.

$$\forall c_i, c_j \in C : Teacher(c_i) = Teacher(c_j) \wedge c_i \neq c_j \Rightarrow y_{c_i} \neq y_{c_j}$$

Instead of considering it as a hard constraint, the maximum number of slots has been implemented as the objective function f_{CTT}, which represents the excess of the number of slots needed. As for f_{CAP}, f_{CTT} has to be minimized. f_{CTT} is defined in Expression 4.14 where $MaxNbSlots$ is the maximum number of slots that can be used.

$$f_{CTT} = \max_{c \in C}(y_c - MaxNbSlots)^+ \tag{4.14}$$

When both problems are simultaneously solved, the variables and the constraints are the same as in CAP and CTT. The objective function is the linear function $f_{CAP\&CTT}$ defined in Expression 4.15 that combines both objectives. As the number of slots is a hard constraint modeled as a soft constraint, γ has to be much higher than α or β.

$$\begin{aligned} f_{CAP\&CTT} &= \alpha \cdot \max_{s \in S}\left(RankGap_\mu(s)\right) \\ &+ \beta \cdot \sum_{s \in S}\left(RankGap_\mu(s)\right) \\ &+ \gamma \cdot \max_{c \in C}(y_c - MaxNbSlots)^+ \end{aligned} \tag{4.15}$$

4.6.1 The CSOP algorithm for the CAP&CTT

As both problems are NP-hard, the optimal solution to the combined problem, CTT&CAP, cannot be found in a reasonable amount of time. We have therefore implemented an algorithm that uses a CSOP solver with a timeout to find a solution to the CTT or to the CAP or to both problems simultaneously. The algorithm is presented in Algorithm 10.

In the first stage of this algorithm, lines 1 to 6, the courses that don't have enough students are canceled. The criterion used to cancel a course is the number of students allocated to it. If there is at least one course with less than a minimum number of students Min_{st}, then the smallest course is canceled and the new CAP, with one course less, is solved again. After canceling a course, disappointed students may fill another course that was also below the minimum in the first step, that is why the new CAP is solved again and several courses are never canceled simultaneously. The CTT problem is then solved with the CAP allocation considered as a constraint.

If the CTT solution found in the first step needs more slots than available, the second stage of this algorithm, lines 7 to 13, is executed. The CAP&CTT are then simultaneously solved.

After that, if more slots than available are still needed, additional compatibility constraints are added to the CAP and the second stage starts again in line 7, with the new compatibility constraints and the courses already canceled.

In lines 10 to 12, the compatibility constraints that should be added are selected. One course and one slot not assigned to this course are selected. The combinations of this course and the courses assigned to this slot are forbidden, so that this course can be assigned to this slot. The selection of the course is based on the slot load: the slot of the selected course is the slot that is assigned to the smallest number of courses. The selected slot is the one that minimizes the number of students affected by the new constraints.

Algorithm 10: The CSOP approach

Input: List of courses C; List of students S; Ranking $Rank_s(c), \forall c \in C, s \in S$; Capacity $K(c), \forall c \in C$

Output: Allocation μ, Timetable ν

1 Solve CAP;
2 **while** $min_{c \in C}(Card(\mu(c))) < Min_{st}$ **do**
3 Select $c \in \{c \in C : Card(\mu(c)) = min_{c \in C}(Card(\mu(c)))\}$
4 $K(c) \leftarrow 0$;
5 Solve CAP ;
6 Solve CTT for the last allocation μ found in the previous step;
7 **if** *Nb slots needed > Nb slots available* **then**
8 Solve CAP & CTT ;
9 **if** *Nb slots needed > Nb slots available* **then**
  `/* Add CAP constraints to reduce the number of slots needed */`
10 Select $c \in C$ so that $Card(\nu^{-1}(\nu(c)))$ is minimal ;
11 Select $ts \in TS$ so that $ts \neq \nu(c)$ and $Card(\{s \in \mu(c) : s \in \mu(\nu^{-1}(ts))\}$ is minimal;
12 Forbid the combinations $(c, c_i), \forall i \in \nu^{-1}(ts)$;
13 Back to 7;

4.7 The ACO approach for the CAP&CTT

Algorithm 11 describes the steps of the ACO approach. The courses are canceled and the CAP is solved in the same way as in the first stage of Algorithm 10, lines 1 to 6. In the second stage, instead of combining both problems to find a complete solution, the ants are used to find a feasible timetable first, line 11. This timetable is then a constraint for the CAP since clashes are forbidden, line 14, and then the CAP is solved as a CSOP.

At the end of an iteration, the best solution will correspond to the best solution found by all the ants of this iteration. At the end of the algorithm, the global best solution will correspond to the best solution of the solutions of the iterations.

At the end of each iteration, the pheromones evaporate partially and both, the best solution of the iteration and the global best solution, deposit pheromones on the corresponding timetable. Algorithm 12 contains the description of the evaporation and deposition process where $\rho \in (0, 1)$ is the evaporation rate and φ is the weight for the solution cost.

Algorithm 11: The ACO approach

Input: List of courses C; List of students S; Ranking $Rank_s(c), \forall c \in C, s \in S$; Capacity $K(c), \forall c \in C$; Number of iterations $NbIter$; Number of ants $NbAnts$
Output: Allocation μ, Timetable ν

```
Solve CAP-CSOP ;
while min_{c∈C}(Card(μ(c))) < Min_st do
    Select c ∈ {c ∈ C : Card(μ(c)) = min_{c∈C}(Card(μ(c)))};
    K(c) ← 0;
    Solve CAP-CSOP → Allocation μ;
Solve CTT-CSOP for the last allocation μ found;
if Nb slots needed > Nb slots available then
    Initialize pheromones;
    for i ← 1, NbIter do
        for a ← 1, NbAnts do
            Solve CTT-Ant → Timetable ν;
            for {c_i, c_j} ∈ C do
                if ν(c_i) = ν(c_j) then
                    Forbid combination (c_i, c_j);
            Solve CAP-CSOP;
            Update best solution of the iteration;
        Update pheromones for the best solution of the iteration;
        Update global best solution;
        Update pheromones for the global best solution;
```

4.8 Experimentation: Greedy versus CSOP for the CAP

Ten sets of data have been generated for the ranking of 50 students over 10 courses. This ranking is based on the distribution of the preferences of the students over the courses given in Table 4.2: the courses with a higher probability will have a higher probability to be among the first choices of students. For example, course 4 has a probability of 7% to be the first choice of a student.

All courses have a capacity of fifteen seats, $\forall c \in C : K(c) = 15$. Three courses are allocated to each student: $\gamma = 3$. The generated ranking is a strict ranking, which means that even if a

Algorithm 12: Pheromones update in the ACO approach

Input: Timetable solution ν; Pheromones τ; Timetable solution cost $Cost$
Output: Updated pheromones τ

```
for c ∈ C do
    for ts ∈ TS do
        τ_c^ts = (1 − ρ) · τ_c^ts                    // evaporation
    τ_c^{ν(c)} = τ_c^{ν(c)} + φ/Cost                 // deposition
```

Course	1	2	3	4	5	6	7	8	9	10
Probability	5%	5%	6%	7%	10%	10%	12%	12%	15%	18%

Table 4.2: Experimentation. Distribution of preferences of the students over courses.

	CGA	CSOP 10 sec	CSOP 1 min	CSOP Best known	Best
(TSG)	226 ± 19	217 ± 21	215 ± 21	211 ± 20	CSOP
(WSG)	12 ± 3	8 ± 1	8 ± 1	7 ± 1	CSOP

Table 4.3: 50 students, 10 datasets - Comparison CGA-CSOP, Mean ± Standard Deviation of the two metrics (TSG) and (WSG).

student s is assigned to his first three choices c_1, c_2 and c_3, the position of this courses for s will be $P_s(c_1) = 0$, $P_s(c_2) = 1$, and $P_s(c_3) = 2$. Therefore for any allocation μ and any student $s \in S$ we have:

$$SatGap_\mu(s) \geq 3 \qquad \text{(TSG)} \geq 150 \quad \text{and} \quad 24 \geq \text{(WSG)} \geq 3.$$

To solve the CSOP, we have used Gecode, [103, 131]. The computation was done with one single processor on an Intel® Core™ i5-3320M CPU 2.60 GHz.

4.8.1 The results

Table 4.3 contains the results. For CGA, the solution is found in less than 1 second. For CSOP, the table contains the results after 10 seconds of computation, after 1 minute and the best known solution.

Regarding (TSG), CSOP is on average 4% better than CGA after 10 seconds, and 6.6% better for the best known solution. The ten charts in Figure 4.1 correspond to the ten datasets analyzed and contain the value of (TSG) for CGA, as this algorithm finds only one solution, the value of (TSG) doesn't change. The charts also contain the results for the CSOP method where we can see for each solution found, the value of (TSG) and the computation time needed to find it.

Regarding (WSG), the first solution obtained with CSOP is always better than with CGA. After 10 seconds, CSOP is 33% better and even 42% better for the best known solution.

Figure 4.2 compares the value of (TSG) with CGA and CSOP for the first solution, after 10 seconds, 30 seconds, 1 minute and 15 minutes, and includes also the best known result for the CSOP. The first solution obtained with CSOP is not always better than the CGA solution, but after 10 seconds of computation, CSOP outperforms CGA in all datasets.

Additional simulations were done with ten datasets of 150 students in order to analyze the possible loss of effectiveness for bigger datasets. Figure 4.3 and Table 4.4 compare the results

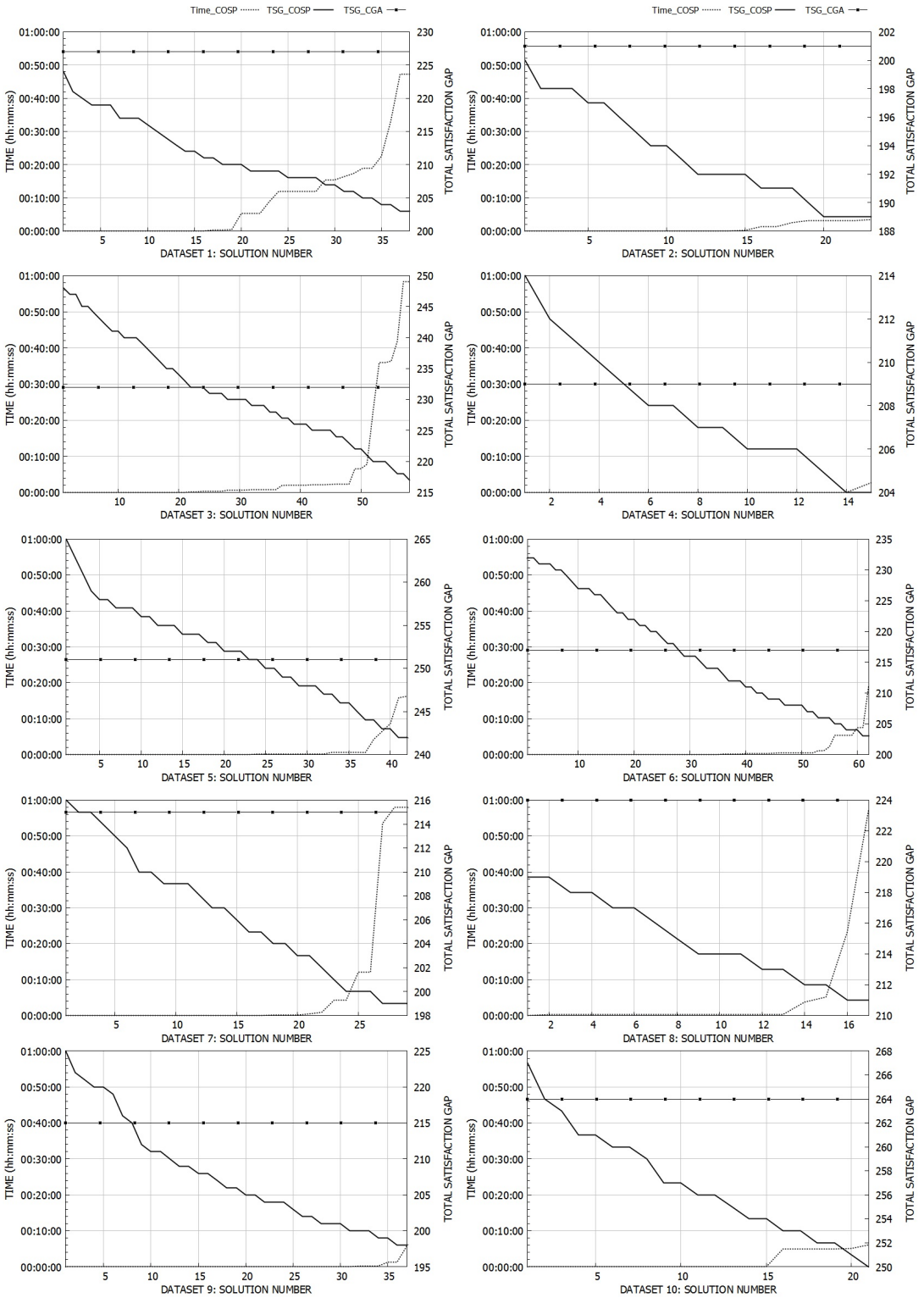

Figure 4.1: 50 students - Results: Value of the metric (TSG) in the different solutions found by CSOP and time when these solutions are found.

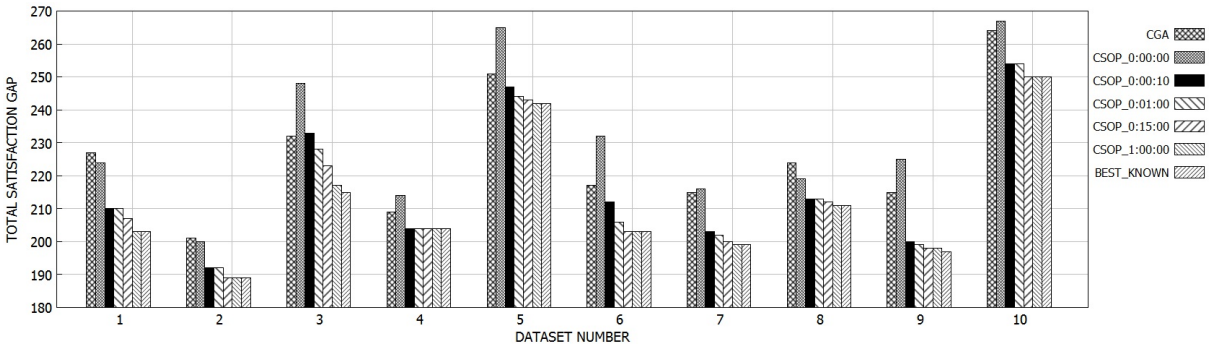

Figure 4.2: 50 students - (TSG) value for CGA and CSOP for each of the ten datasets in the first solution and after 10 seconds, 1 minute, 15 minutes and 1 hour of computation.

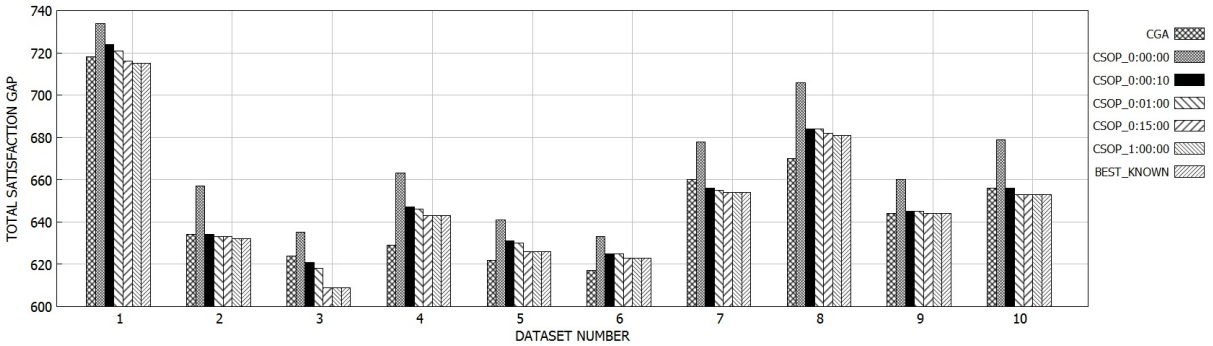

Figure 4.3: 150 students - (TSG) value for CGA and CSOP for each of the ten datasets in the first solution and after 10 seconds, 1 minute, 15 minutes and 1 hour of computation.

obtained for (TSG) with CGA and the CSOP method for the first solution, after 10 seconds, 1 minute, 15 minutes and 1 hour, and also include the best known result with the CSOP. Regarding (TSG), neither algorithm outperforms the other: CSOP is better in five sets and CGA is better in the other five sets. Regarding (WSG), CSOP always outperforms CGA and on average the results are 44% better.

	CGA	CSOP 10 sec	CSOP 15 min	CSOP Best known	Best
(TSG)	647 ± 31	652 ± 31	648 ± 31	648 ± 31	CGA
(WSG)	18 ± 5	10 ± 1	10 ± 1	10 ± 1	CSOP

Table 4.4: 150 students, 10 datasets - Comparison CGA-CSOP, Mean $\pm$ Standard Deviation of the two metrics (TSG) and (WSG).

dataset	**1**	**2**	**3**
Number of courses offered	20	19	18
Number of students	326	279	271
Nb of courses with rank 0 per student	3.6 ± 1.4	3.9 ± 1.7	3.3 ± 1.2
Nb of courses with rank 0 or 1 per student	5.7 ± 2.3	6.2 ± 2.6	5.2 ± 1.7

Table 4.5: Size of the datasets. The number of courses is given with the average ± the standard deviation.

4.9 Experimentation: CSOP versus ACO for the CAP&CTT

The three datasets used in this section are real data from EHL where elective courses are offered to students for their last semester of the Bachelor program. The following constraints have to be satisfied:

- Each student has to be allocated to 2 courses among those that have been offered.
- Each course has to be scheduled on one weekday (Monday to Friday) avoiding clashes in the students' timetable, so there are 5 timeslots.
- Each course has a capacity of 40 students.
- Some courses are similar. A student cannot be assigned to two courses that are similar. Those pairs of courses are not compatible.

The ranking of the courses by the students contains no gap and at least 2 courses are the first choice of each student, which means $\forall s \in S$:

- $\exists c_i, c_j \in C : c_i \neq c_j \wedge Rank(c_i) = 0 \wedge Rank(c_j) = 0$
- $\forall c_i \in C : Rank(c_i) > 0 \Rightarrow \exists c_j \in C : Rank(c_j) = Rank(c_i) - 1$

Table 4.5 contains for each dataset the number of students and the number of courses offered. It contains also some data about the distribution of the students' ranking. We can see that the dataset 3 may be more difficult since the average number of courses with a rank 0 or 1 is much smaller than in the other datasets. Considering the same indicators the second dataset should be easier.

Each semester, the offer exceeds the demand, which means that more courses than needed are proposed to the students. The courses with the lowest demand are not opened, the students' ranking is used to identify those courses. The number of courses that are canceled depends on the students' ranking. Table 4.6 contains the average number of courses with a rank 0 or 1 once the least popular courses are canceled; this result is provided by both approaches, CSOP and ACO. The most difficult dataset seems to be still the third one. Nevertheless the impact of

dataset	1	2	3
Number of courses canceled	2	3	3
Nb of courses with rank 0 per student	3.4 ± 1.4	3.4 ± 1.5	3.1 ± 1.1
Nb of courses with rank 0 or 1 per student	5.4 ± 2.2	5.4 ± 2.3	4.8 ± 1.7

Table 4.6: Data after canceling courses. The number of courses is given with the average ± the standard deviation.

closing the courses is more important on dataset 2 than on the dataset 3. datasets 1 and 2 are similar for those indicators, even if 1 more course is closed in dataset 2.

The computation was done on an Intel® Core™ i5-5300U CPU 2.30 GHz. Gecode, [131], was used to solve the CSOP models: CAP, CTT, CAP&CTT. As the planification requires time efficiency and it must be performed quickly, the computation time for each launch of Gecode was limited to 10 seconds in the first stage and to 2 minutes in the second stage of the CSOP algorithm. It was limited to 5 seconds for each ant in the ACO algorithm.

To solve the CTT problem with ACO, we have used a colony of 20 ants. With some CTT solutions built by the ants, there was no solution to the CAP. The limit was not a number of iterations, but a time limit of 14 minutes.

The total computation time for the CSOP approach is between 8 and 11 minutes for the three datasets.

4.9.1 Results for the CSOP approach

Figure 4.4 contains the results for the three datasets for the CSOP approach. The charts contain the value of the two metrics (TRG) and (WRG) and the number of slots needed for the best solution found each time Gecode was launched. The CTT is not solved in the first stage of the algorithm, the number of slots is therefore not available in the first solutions.

When a course is canceled just before launching Gecode (Algorithm 10, line 4), a square is plotted on the best solution found by Gecode right after (Algorithm 10, line 5). When constraints are added to the CAP just before launching Gecode (Algorithm 10, lines 10-12), a circle is plotted on the best solution found by Gecode right after (Algorithm 10, line 8).

If we compare (TRG) for the first solution found, we can see that dataset 2 has a better value. This can be compared with the conclusions from Section 4.9 where this dataset seemed to be easier due to the higher average number of courses with rank 0 per student. In the same way the dataset 3 has a worse value for (TRG) and it has also a smaller average number of courses with rank 0 per student.

In the three charts, we can clearly see the impact of canceling a course on (TRG); in dataset 2, the second chart, this situation is worse since (TRG) goes from 25 to 87 (44 to 64 in dataset 1 and 50 to 89 in dataset 3). This happens also for (WRG), except for dataset 1. The smaller impact on dataset 1 can be explained by the fact that there are some courses that are not popular and the fact that those courses are not offered does not affect the students' satisfaction

In the three datasets, new constraints had to be added to the CAP in order to find a feasible timetable, since in all sets 8 slots were initially needed. Adding a new constraint has also an impact on (TRG), but much less than canceling a course and has almost no impact on (WRG). In dataset 2, after adding a second constraint, (WRG) is even better than before; this is probably due to the fact that the computation time is limited and in the previous launch of Gecode, the best solution was not found due to timeout.

4.9.2 Results for the ACO approach

Figure 4.5 contains the results for the three datasets for the ACO approach. The charts contain the value of the two metrics (TRG) and (WRG) for the solution found by each ant for each dataset; this solution includes the CTT solution found by the ant itself and the best solution found by Gecode for the CAP. The number of slots used in the timetable is not included since the ants build only feasible timetables with 5 slots.

In those charts, we can clearly see how the ACO approach converges to the best solution, which is found in iteration 10 or 11 in the three datasets after a computation time of around 8-10 minutes.

4.9.3 Comparison of CSOP and ACO

Table 4.7 contains the indicators for the best solution found for each dataset and Table 4.8 gives the detail of the distribution of (WRG) for all students, $(WRG) = 0$ being the best situation for a student.

In Table 4.7, we can see that for the first dataset, the results for both methods are identical. In the second dataset, ACO provides a solution that is better for both metrics. In the third dataset, even if the result (TRG) is worse, as the weight used for (WRG) is $\alpha = 100$ and the weight used for (TRG) is $\beta = 1$, the objective function described in Expression 4.12 is much better with ACO than with CSOP.

In Table 4.8, we can see that in the three datasets with both approaches, more than 70% of the students have $RankGap = 0$, which means that they are fully satisfied since they are allocated to 2 courses that belong to their first choices.

In the first dataset, even if the results for (WRG) and (TRG) are the same for both approaches, the ACO result is more interesting for the worst-off students since there are less students with $RankGap = (WRG)$. Nevertheless, this happened by chance since there is no incentive in the objective function for this situation.

4.9.4 Computation time

Table 4.9 contains the computation times. The *total* computation time corresponds to the total time used for the computation. *Feasible* solution gives the time needed to find a feasible solution with 5 slots in the timetable.

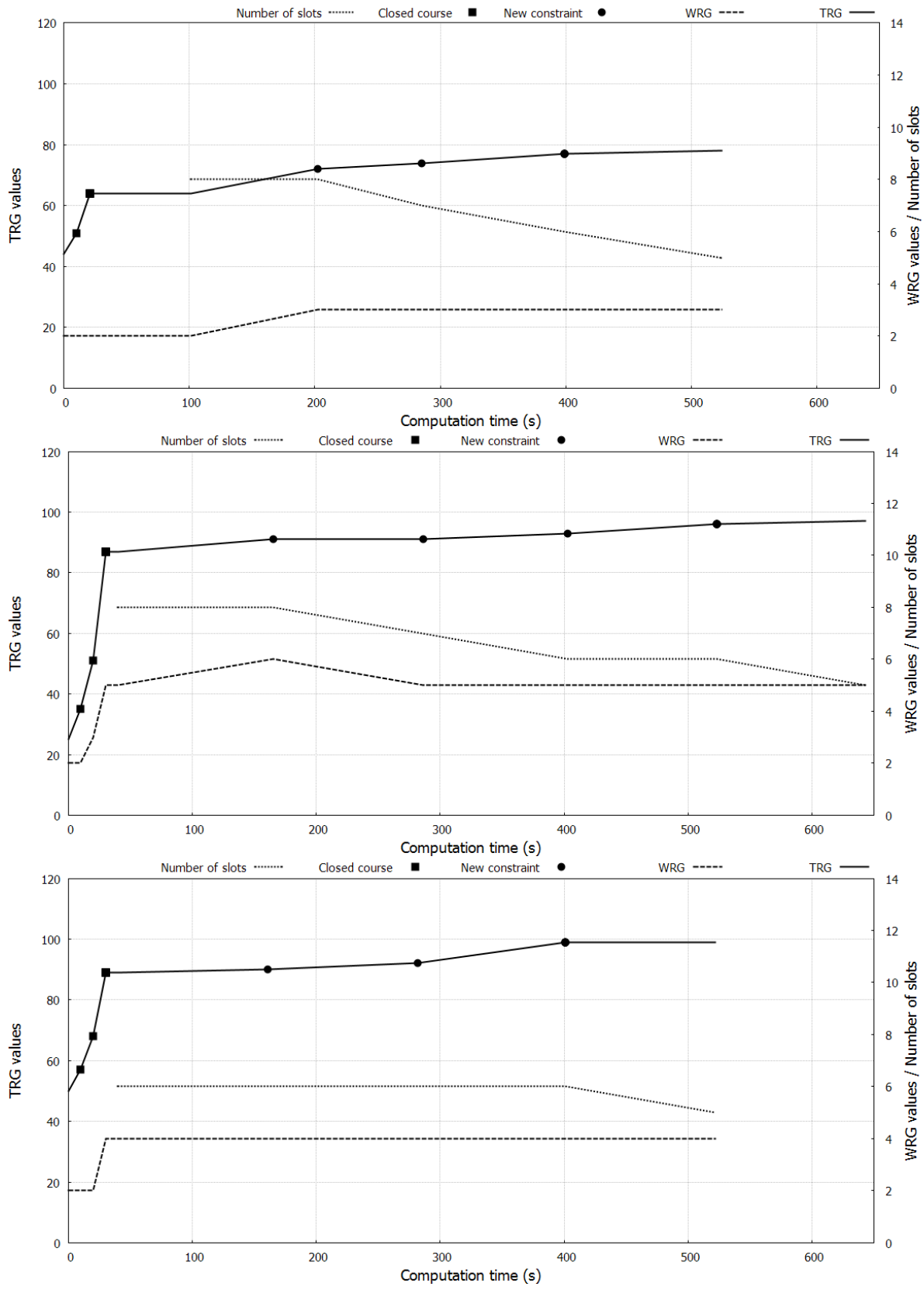

Figure 4.4: Value of the metrics (TRG) and (WRG) and number of slots needed in the different solutions found by CSOP.

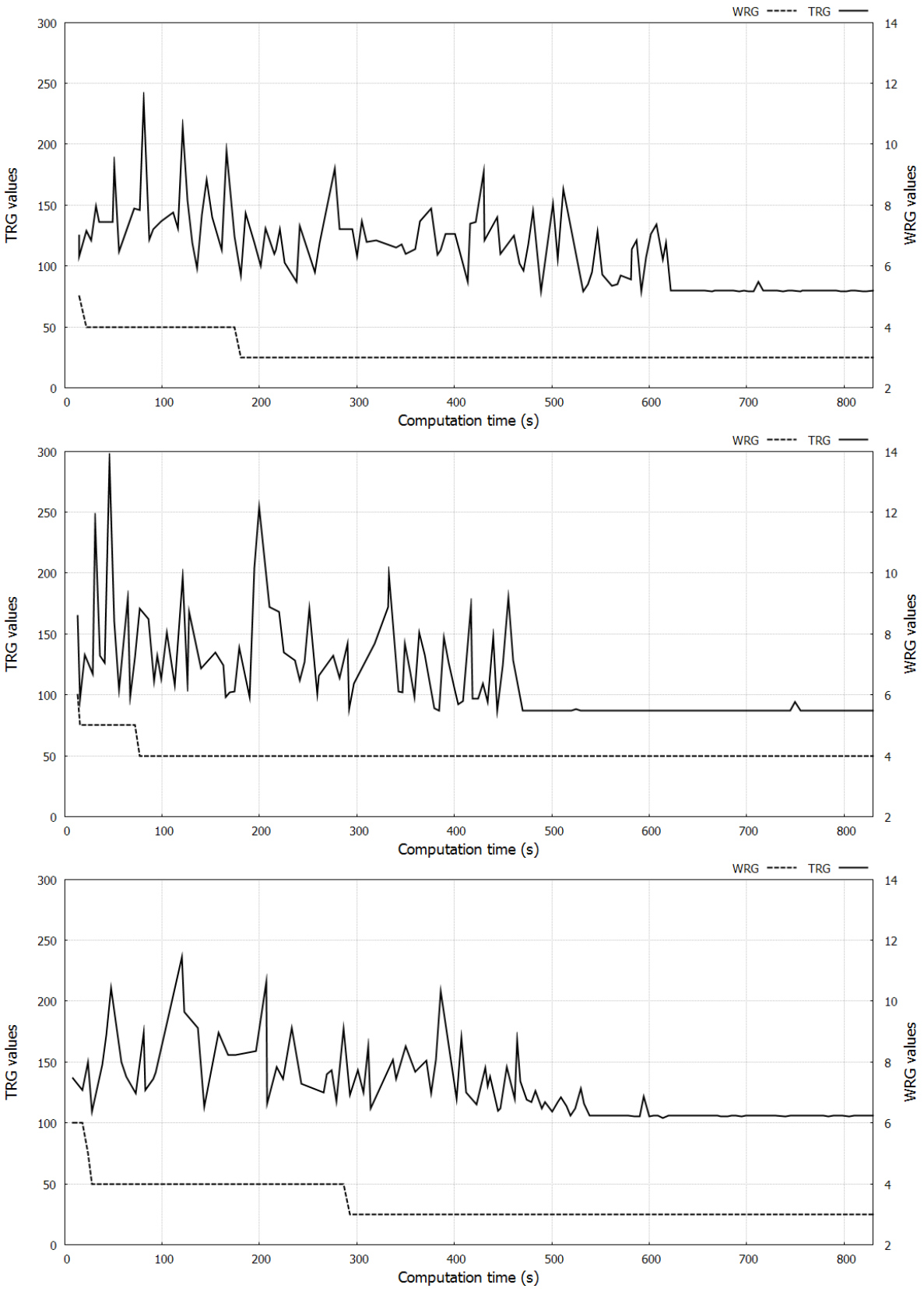

Figure 4.5: Value of the metrics (TRG) and (WRG) in the different solutions found by ACO.

	dataset 1		dataset 2		dataset 3	
	CSOP	ACO	CSOP	ACO	CSOP	ACO
(TRG)	78	78	97	87	99	104
(WRG)	3	3	5	4	4	3

Table 4.7: Values of the two metrics (TRG) and (WRG) for the CSOP and the ACO approaches.

Rank Gap value	**Number of students**					
	dataset 1		**dataset 2**		**dataset 3**	
	CSOP	ACO	CSOP	ACO	CSOP	ACO
0	265	263	223	224	200	200
1	49	51	34	34	48	45
2	7	9	9	12	19	19
3	5	3	8	7	3	7
4	0	0	4	2	1	0
5	0	0	1	0	0	0

Table 4.8: Distribution of the *RankGap* value for the students for the CSOP and the ACO approaches.

As in the ACO approach, the number of slots is a hard constraint, all found solutions are feasible. If we consider that a *good* solution is the one that minimizes (WRG), depending on the dataset, a good solution is found within 1 to 5 minutes. The *best* solution is found within 4 to 10 minutes. Regarding the CSOP approach, the number of slots is a soft constraint as are the metrics (WRG) and (TRG). Therefore all these computation times are the same since the computation is stopped as soon as a feasible solution is found.

Nevertheless, the computation time was not a criterion in our investigation. Those times can certainly be reduced if we adjust the timeout for each launch of Gecode.

	dataset 1		dataset 2		dataset 3	
	CSOP	ACO	CSOP	ACO	CSOP	ACO
Total	8:43	14:00	10:43	14:00	8:42	14:00
Feasible	8:43	0:00	10:43	0:00	8:42	0:00
Good	8:43	2:46	10:43	1:17	8:42	4:39
Best	8:43	7:56	10:43	4:42	8:42	9:34

Table 4.9: Computation time for the CSOP and the ACO approaches [mm:ss].

4.10 Conclusions

In this chapter, we have analyzed and solved two different types of problem. The first type is a simple Course Allocation Problem, and the second type combines a Course Allocation Problem with a Course Timetabling Problem.

For the simple CAP, we have compared two different approaches to solve the problem of allocating several courses to students, where each student gives his preferences with a strict ranking of the available courses. Two metrics have been used to quantify the quality of a solution: the Total Satisfaction Gap (TSG) analyzes the average level of satisfaction of the students and the Worst Satisfaction Gap (WSG) corresponds to the level of satisfaction of the worst off. The first approach, the Course Greedy Algorithm (CGA), allocates a course to the students who ranked this course first, and if seats are still available, to those who ranked it second and so on. This mechanism is fairer than a simple First-come First-served mechanism. The second approach is based on a Constraint Satisfaction Optimization Problem (CSOP) and uses the solver Gecode to optimize the value of both metrics. For small instances, the results for both metrics are very quickly much better than with CGA. When the number of students increases, the benefit regarding (WSG) is even more important than for smaller instances. However, the computation time needed to obtain good results for (TSG) increases, and the benefit of this approach is less important than for smaller instances since (TSG) with CSOP is similar to (TSG) with CGA with 150 students. Nevertheless the allocations obtained with CSOP are much fairer for students than the allocation obtained with CGA.

For the combined CAP&CTT, we have presented two approaches to solve a problem that combines a timetabling problem with a multi-course allocation problem whose objective is to maximize the satisfaction of students. The first approach uses a CSOP solver to find the solutions to both problems simultaneously and to each problem individually. The second approach uses the same CSOP solver for the courses allocation, but an ACO algorithm to solve the timetabling problem. Both approaches provide very good results in a reasonable computation time. The ACO approach is nevertheless much faster in finding good solutions. Moreover all the solutions provided by the ACO approach are feasible as the timetable constraints are not considered as soft, but as hard constraints. Those approaches are currently used at EHL to allocate the seats in the courses and to guarantee that this allocation is compatible with the timetable constraints.

Chapter 5

Comparison of Three Metaheuristics to Handle Many-objective RAP

In this chapter, ACO and HS are adapted to a many-objective RAP. Their performance is compared to the classic and well-known NSGA-II that is a GA adapted to many-objective problems, the Non-dominated Sorting Genetic Algorithm II. The data used to compare their performance come from Canton de Vaud and is the same as the one used in chapter 3.

5.1 Introduction

In this chapter, the problem of the educational system in Canton de Vaud presented in chapter 3 is modeled as a real many-objective optimization problem (MOOP). The seven objectives of this MOOP revolve around the number of teaching hours, the number of teachers and the pedagogical goal. Three bio-inspired approaches are applied to this problem and compared, an ACO algorithm and a HS algorithm adapted to the MOOP, and the well-known NSGA-II, the Non-dominated Sorting Genetic Algorithm II. The three of them use the crowding distance value to sort the solutions, together with the successive Pareto fronts in NSGA-II and HS. This sorting is then used to guide the search: the best solutions deposit more pheromones in the ACO approach, the worst solutions are removed from the population in NSGA-II and from the harmony memory in the HS approach.

The rest of the chapter is structured as follows. Section 5.2 describes the school's problem in Canton de Vaud. Section 5.3 presents the model for this problem. Sections 5.4, 5.5, and 5.6 explain the three approaches, ACO, HS and NSGA-II. Section 5.7 presents the results and the two hypervolume methods used to compare the performance of the three approaches. Section 5.8 contains the conclusions and presents future work.

5.2 Description of the Problem

In secondary schools in Switzerland, students are assigned to main classes and all students of a main class attend most courses together, except elective courses and courses with different levels, hereinafter referred to as multi-level courses [104]. For the multi-level courses, students are split according to their level and then grouped with students of other main classes who have the same level. The same teacher can teach any level of a course for which he possesses the required skills. In Canton de Vaud, three courses have different levels: French, German and Mathematics.

If we consider only the timetabling problem that exists in all schools, the best solution for the allocation of students is to assign students with the same profile to the same classes. The splitting-grouping procedure is thus minimized or even not needed and scheduling is easier.

However the allocation of students to main classes must satisfy a pedagogical objective that is not in line with the timetabling problem. The composition of a main class must be as mixed as possible considering the different levels. The splitting-grouping phase thus becomes mandatory and implies that the timetables of two or more main classes have to be compatible for multi-level the courses. Furthermore, the compatibility of timetables for a course implies that more teachers are needed, since two or more lessons of the same course take place simultaneously. If timetables are not compatible, the lessons for the course have fewer students.

In this problem, we have three types of objectives and seven different objectives:

- The pedagogical objective, which establishes that classes should be as mixed as possible.
- The number of teaching hours for each of the three multi-level courses should be minimized.
- The number of teachers, which should be as low as possible for each of the three multi-level courses. Teachers are difficult to find, especially when they teach just a few hours.

If, for a multi-level course, the slot in the timetable is the same for all main classes, the number of hours is minimized since all students are available at that moment for this course. The main classes can then be as mixed as possible, but as all lessons for this course take place simultaneously, the number of teachers needed is not minimized. In order to minimize the number of teachers, the lessons for this course should never have the same slots in the timetable and the students in a main class should have the same level in order to minimize the number of lessons.

5.3 Problem Statement

This allocation problem is a many-objective resource allocation problem (MORAP) and is modeled in this chapter as a constraint satisfaction optimization problem (CSOP).

A CSOP for the MORAP is described with four sets $(X; D; C; F)$, where X contains the variables that have to be defined, the domain D contains the possible values that may take the

variables and C is the set of constraints that must be satisfied by an assignment of values to the variables so that this assignment is a feasible solution of the problem. F is the set of the objectives of the MORAP, each objective being a fitness function that represents a quantitative measure of the quality of an assignment for this objective.

The variables of the CSOP in the MORAP are $X = \{X_1, \ldots, X_n\}$, where n is the number of students. The domain is the same for all variables, $D = \{d_1, \ldots, d_m\}$, where m is the number of main classes and d_i is an identifier for the i-th main class. The constraint of the CSOP is the capacity of a main class, i.e. no more than D_{max} students per class:

$$Card(\{X_i \in X : X_i = d_j\}) \leq D_{max}$$

where $Card(\cdot)$ denotes set cardinality.

An allocation $\mu = (x_1, x_2, \ldots, x_n)$ is defined as an assignment of a value $x_i \in D$ to each variable $X_i \in X$, that is the assignment of one main class to each student. For $s \in S$, $\mu(s)$ is the main class allocated to s. Hereinafter S is the set of students, D the set of main classes, and C the set of courses with levels.

5.3.1 The Pedagogical Objective

The pedagogical objective can be evaluated comparing the diversity of the different main classes. If the diversity is similar, then it is maximized in all main classes. To evaluate this similarity, each level in every course is considered as a category $k \in K$, e.g. students with level 1 in French belong to the corresponding category k_F^1. Therefore a student may belong to several categories, the categories of a student $s \in S$ are $SK(s)$. Each course $c \in C$ has also several categories $CK(c)$, each of them corresponding to a level.

For an allocation μ, the number of students in a main class $d \in D$ belonging to a category $k \in K$ is given by Expression (5.1). The optimal similarity for a category $k \in K$ is reached if all main classes have the same number of students who belong to this category, this optimal number is expressed in Expression (5.2):

$$DN_\mu(d, k) = Card(\{s \in S : x_s = d \wedge k \in SK(s)\}) \tag{5.1}$$

$$\overline{DN}(k) = \frac{Card(\{s \in S : k \in SK(s)\})}{m} \tag{5.2}$$

The similarity between classes - the first objective of the MORAP - can then be evaluated as in Expression (5.3).

$$f_0(\mu) = \sum_{k \in K} \sum_{d \in D} |DN_\mu(d, k) - \overline{DN}(k)| \tag{5.3}$$

5.3.2 The Number of Lessons and Teachers

The number of lessons and the number of teachers are determined by the number of occurrences of a course and how those occurrences are scheduled in the timetable. In order to compute those values, we define here the concept of groups of main classes.

The main classes are classified into groups: one group has a maximum of m main classes. Classes within the same group must have the same timetable for the multi-level courses. Students of this group are split and grouped to attend the lessons corresponding to their level.

There are several possible groupings of classes. For example, for $m = 2$, we have two groupings: either both classes are in the same group, so there is only 1 group, or they are in different groups yielding 2 groups.

For an allocation μ, a category k and a group g, the number of lessons is given by Expression (5.4), that is the number of students with a given level who belong to this group divided by the maximum number of students in a class, i.e.

$$L_{k,g}(\mu) = \left\lceil \frac{Card(\{s \in S : k \in KS(s) \wedge X_s \in g\})}{D_{max}} \right\rceil . \tag{5.4}$$

Therefore, the number of lessons for a course $c \in C$ is shown in Expression (5.5). As each group has its own slots for a course, the total number of teachers corresponds to the maximum number of lessons that take place simultaneously, which is given by Expression (5.6).

$$f_c^l(\mu) = \sum_g \sum_{k \in CK(c)} L_{k,g}(\mu) \tag{5.5}$$

$$f_c^t(\mu) = \max_g \left(\sum_{k \in CK(c)} L_{k,g}(\mu) \right) \tag{5.6}$$

The objectives of the MORAP are thus to minimize f_0 and f_c^l and f_c^t for all $c \in C$.

In the next three sections, the three approches are described: ACO, HS and NSGA-II.

5.4 The ACO Approach

For the MORAP presented in Section 5.3, an ant builds a solution by assigning a main class to a student at each step. At the end of each iteration, solutions built by the ant colony are compared by using the fitness functions. All non-dominated solutions of the iteration at hand deposit pheromones.

Every possible assignment $(s, d) \in (S, D)$ has, thus, a pheromone quantity $\tau_{s,d}$. At the beginning of the optimization process, pheromones are initialized to a value τ_{init}. At the end

of each iteration, they are reduced by the evaporation rate $\rho \in]0,1[$, and reinforced by the non-dominated solutions of the iteration, as in Expression (5.7). The reinforcement $\Delta_{s,d}$ depends on the crowding distance of the solution: the larger the crowding distance of the solution, the more pheromone is laid on the corresponding assignments:

$$\tau_{s,d} = \begin{cases} \rho \cdot \tau_{s,d} + \Delta_{s,d} & \text{if } (s,d) \text{ belongs to a non-dominated solution} \\ \rho \cdot \tau_{s,d} & \text{otherwise} \end{cases} \tag{5.7}$$

The visibility represents the availability of seats in the different classes. If a class $d \in D$ is full, its visibility η_d is null. When an ant builds a solution, this visibility is updated at each step (that is after assigning each student) as in Expression (5.8).

$$\eta_d = \begin{cases} 1 & \text{if there are seats available in } d \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

When the ant builds the solution, at each step one student s is considered and the probability to assign s to d is given by Expression (5.9).

$$p_{s,d} = \frac{\tau_{s,d} \cdot \eta_d}{\sum_{d_i \in D} \tau_{s,d_i} \cdot \eta_{d_i}} \tag{5.9}$$

Algorithm 13 summarizes the steps needed to find the approximation of the Pareto front. Two sets of solutions are used: Set_{best} contains the solutions found in the iteration that are not dominated by any other solution found in this iteration, whereas Set_{global} contains the solutions found by the algorithm that are not dominated by any other solution found in previous iterations. When a set is updated, the updated set keeps all non-dominated solutions.

Algorithm 13: Multi-objective ACO approach

Input:
Output: Set of non-dominated allocations Set_{global}

```
Initialize pheromones
for i ← 1, ..., N do
    Set_best = ∅
    for ant ← 1, ..., A do
        μ_ant = ∅
        for s ∈ S do
            Select d ∈ D with a probability p_{s,d}
            μ_ant ← μ_ant ∪ (s, d)
            Update η_d
        Update Set_best with μ_ant
    Evaporate pheromones
    Lay pheromones for all μ ∈ Set_best
    Update Set_global with Set_best
```

5.5 The HS Approach

In the model presented in Section 5.3, a harmony is a solution of the class composition problem, each note is thus an allocation of a student to a main class and each musician is in charge of allocating one student.

Algorithm 14 summarizes the steps of the HS approach used for the MORAP problem. The generation of harmonies is performed N times, the initial HM is composed of random allocations of students to main classes (line 1).

Algorithm 14: Multi-objective HS approach

```
   Input:
   Output: HM
1  Populate HM with HM_size random allocations for i ← 1, ..., N do
2      for s ∈ S do
3          if rand(0,1) < HMCR then
4              Select randomly μ_s ∈ HM → d_s = μ_s(s)
5              if rand(0,1) < PAR then
6                  d_s ← d_s + 1 with a probability of 0.5
7                  d_s ← d_s − 1 with a probability of 0.5
8          else
9              Select randomly d_s ∈ D
10         if Card(d_s) < D_max then
11             μ ← μ ∪ (s, d_s)
12     Allocate randomly all non-allocated students to non-full main classes
13     Update μ with the random allocations
14     if μ dominates the worst solution μ_w ∈ HM then
15         Replace μ_w by μ in HM
```

For the composition of a new harmony each musician selects an existing note in HM or generates a random note, depending on the value of the parameter Harmony Memory Considering Rate (HMCR), which ranges from 0 to 1 (lines 3 to 7). A high value of HMCR means a high probability to select an existing note in HM, a small value of HMCR means a high probability to generate a random note. In addition to HMCR a second parameter called Pitch Adjusting Rate (PAR) is used to avoid local minima. With a probability PAR, the note selected by a musician is changed to the closest upper or lower note (lines 5 to 7).

In the model presented in Section 5.3, the class allocated to a student is selected among the classes he is assigned to in the different solutions stored in HM with a probability HMCR and is randomly selected with a probability 1-HMCR. If we assume that this class is d, it is then changed to $d+1$ with a probability PAR/2 and to $d-1$ with a probability PAR/2.

If a student cannot be allocated to the selected class, because it is full, he stays unallocated until the end of the composition. Unallocated students are then allocated uniformly at random to classes that are not full (lines 12-13).

This new allocation μ replaces then the worst solution μ_w stored in HM if μ is better. μ is better than μ_w if μ dominates μ_w or if none is dominating the other and the crowding distance of μ is larger than the crowding distance of μ_w. The crowding distance has to be calculated with the solutions that are in the same front as μ and μ_w in HM. The worst solution in HM is the one with the smallest crowding distance that is dominated by the highest number of solutions in HM.

5.6 The NSGA-II Approach

As explained in Section 2.1.2, genetic algorithms are based on the natural selection process. They start with an initial population of P_{size} solutions. Those solutions are combined together in order to create a new population, the offspring population. Both populations, the parent population P_t and the offspring population Q_t, compete then to be part of the next parent population P_{t+1} that includes the best solutions of both of them.

To select the next parent population, parent and offspring solutions of the current iteration are put in a set of solutions $R_t = P_t \cup Q_t$. The non-dominated solutions of R_t are grouped in a front set F_1 and removed from R_t. Step by step, different front sets F_i are created with the non-dominated solutions of the partially emptied R_t. All solutions of a front F_i are better than the solutions of the following fronts F_{i+k}. Within a front, the crowding distance of each solution is calculated, solutions with a higher value are considered to be better. When all the solutions of a front cannot be selected because of the size of the parent population, only the best are selected. Figure 5.1 illustrates the NSGA-II approach.

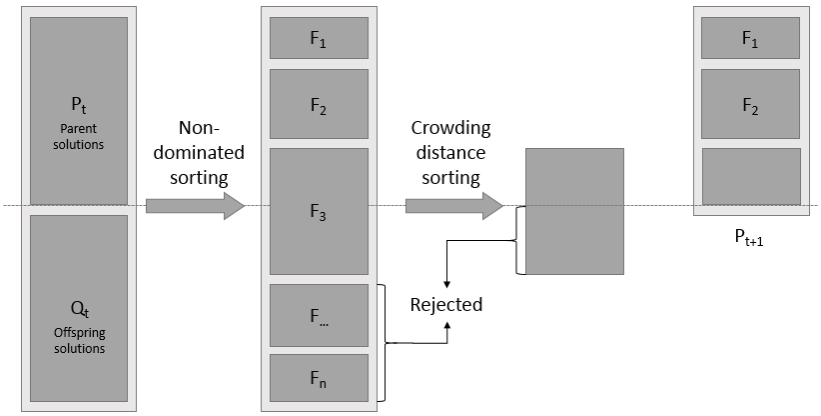

Figure 5.1: NSGA-II approach.

Algorithm 15 describes the steps to select the new parent population P_{t+1} for the next iteration among the parent and the offspring solutions of the current iteration.

Algorithm 16 describes the steps of the NSGA-II approach used for the MORAP problem. The offspring generation is performed N times. For the MORAP presented in Section 5.3, the initial parent population is composed of P_{size} random allocations of students to main classes (line 1).

Algorithm 15: Select the next parent population

Input: Parent and offspring population $R_t = \{\mu_i, i = 1, \ldots, 2 * P_{size}\}$
Output: New parent population P_{t+1}

```
1 P_{t+1} = ∅
2 k = 1
3 while Card(P_{t+1}) < P_size do
4     for μ_i ∈ R_t do
5         if μ_i is non-dominated in R_t and Card(P_{t+1}) < P_size then
6             P_{t+1} ← P_{t+1} ∪ μ_i
7             F_k ← F_k ∪ μ_i
8     R_t ← R_t \ F_k
9     k = k + 1
```

Each solution of the parent population is combined with another solution. The probability of selecting μ_j as the second parent depends on the front of μ_j. If $\mu_j \in F_k$, the smaller k is, the higher the probability is that μ_j is selected as second parent. Better solutions have thus a higher probability to be selected as second parent (line 6).

Algorithm 16: NSGA-II Approach

Input:
Output: Final parent population P_N

```
1  Populate P_0 with P_size random allocations
2  for t ← 1, ..., N do
3      R_t = P_{t-1}
4      for μ_i ∈ P_{t-1} do
5          μ = ∅
6          Select the second parent μ_j ∈ P_{t-1}, j ≠ i
7          for s ∈ S do
8              Select r ∈ {i, j} with a probability of 0.5 each
9              if Card(μ_r(s)) < D_max then
10                 μ ← μ ∪ (s, μ_r(s))
11         Allocate randomly all non-allocated students to non-full classes
12         Update μ with the random allocations
13         Mutate μ with a probability Prob_mutation
14         R_t ← R_t ∪ μ
15     Select the new parent population P_t in R_t
```

A student who is in the same main class in both parents, is allocated to this class in the offspring solution. A student who is in two different classes, is allocated to one of them randomly (lines 7 to 10). If those classes are full and the assignment is thus impossible, the student stays unallocated until the end of the crossover. Unallocated students are then allocated randomly to classes that are not full (line 11).

Each offspring solution generated by a crossover has a probability to mutate. The muta-

Dataset	21-9	21-10	12-9	12-10	34-9	34-10	36-9	36-10
Students	113	137	71	90	78	78	68	66
Main classes	6	8	4	5	5	5	4	4

Table 5.1: Experimentation. Size of the datasets.

tion operation consists in selecting randomly two students allocated to different classes and in swapping their allocation (line 13).

5.7 Experimentation

5.7.1 Datasets

Eight datasets were used to test the performance and compare the three approaches. They contain a list of students with their level in three different courses, French, German and Mathematics. These datasets come from different schools in Switzerland and correspond to the academic period 2015-2016. Table 5.1 contains the number of students and the number of main classes per dataset. The identifier of a dataset indicates the school and the grade, e.g. 21-9 is the dataset for the grade 9 of the school 21.

5.7.2 Parameters

With the NSGA-II model, 80 iterations with a population of 80 solutions were used; the mutation probability was set to 0.1. With the ACO approach, 80 iterations with a colony of 80 ants were used; the evaporation rate ρ was set to 0.9. With the HS model, 6,400 iterations were used with a memory size of 80 harmonies; HMCR was set to 0.7 and PAR was set to 0.1. 100 runs have been applied to the eight datasets with the three approaches. It is important to remark that the above parameters ensure a fair comparison in terms of complexity, as all approaches execute the same number of fitness evaluations. The value of those parameters were set experimentally.

5.7.3 Hypervolume Comparison

Two methods have been used to compare the results of the three approaches in this chapter. The first method is the simple hypervolume comparison (HC). The second method is the smoothing hypervolume comparison (SHC). For the SHC, runs have been grouped into 10 sets of 10 runs and a new Pareto front has been created for each set taking all the non-dominated solutions found in the 10 runs of the set.

As the problem considered in this chapter is a minimization problem, a reference point has to be chosen in order to play the role of the origin. This reference point has to be worse than any other point in the Pareto front. This reference point together with the origin represent a hypercube. As the computation of the hypervolume is complex due to the fact that it has seven

Solution	f_0	f_1^l	f_2^l	f_3^l	f_{total}^l	f_1^t	f_2^t	f_3^t	f_{total}^t
1	12	5	4	4	*13*	5	4	4	*13*
2	12	5	5	5	*15*	3	3	3	*9*
3	12	6	6	6	*18*	2	2	2	*6*
4	28	5	4	5	*14*	3	2	3	*8*
5	28	5	5	4	*14*	3	3	2	*8*
6	34	5	4	4	*13*	3	2	2	*7*
7	60	6	5	6	*17*	2	2	2	*6*
8	64	5	6	6	*17*	2	2	2	*6*
9	66	6	6	5	*17*	2	2	2	*6*
10	86	5	5	6	*16*	2	2	2	*6*
11	92	5	5	5	*15*	2	2	2	*6*

Table 5.2: Non-dominated solutions for a run for the dataset 34-10 with the ACO approach.

dimensions coming from the seven objectives of the problem, a Monte Carlo simulation can be used to estimate it. For each approach proposed in this chapter, a Monte Carlo simulation estimates thus the portion of the hypercube covered by the Pareto set approximation of each run. This portion corresponds to HC.

As the three methods in the benchmark are controlled by stochastic parameters and the result for isolated runs might be strongly biased by local optima, the SHC minimizes this bias by calculating a new approximation of the Pareto front combining the Pareto sets of several runs. The values for SHC should thus be better than those for HC and more significant from a statistical point of view for the comparison of the three algorithmic approaches.

For the estimation of the hypervolume with the Monte Carlo simulation, each value of the reference point for each dataset is the worst value found for each objective among all the non-dominated solutions of the 300 runs, independently for the approach. This reference point is then used to calculate the hypervolume with both methods, HC and SHC, for the three approaches, ACO, HS and NSGA-II. A sampling of 2,000 points has been generated for each dataset and used for the three approaches.

5.7.4 Results and Discussion

As an example of the results, Table 5.2 contains the eleven non-dominated solutions found during one run for the dataset 34-10 with the ACO approach.

As there are seven objectives, the solutions set cannot be plotted in a chart. In order to visualize part of the properties of those sets, the total number of lessons $f_{total}^l = f_1^l + f_2^l + f_3^l$ and that of teachers $f_{total}^t = f_1^t + f_2^t + f_3^t$ have been computed. Figure 5.2 contains the solutions for six datasets. The horizontal axis corresponds to f_{total}^l and the vertical axis corresponds to f_{total}^t. Each circle corresponds to a solution and the size of the circle depends on the value of

		12-9		21-9	
	Approach	100 runs	10 sets	100 runs	10 sets
Nb of runs with	ACO	81	10	9	6
f_0 minimal	HS	76	10	2	2
	NSGA-II	99	10	33	10
f_0, average	ACO	8.6±1.2	N/A	17.6±3.5	13.0 ± 1.3
± std dev.	HS	8.5 ±0.9	N/A	18.1±2.1	14.8 ± 1.8
	NSGA-II	8.02±0.2	N/A	14.3±2.0	N/A

Table 5.3: Comparison of f_0 by ACO, HS and NSGA-II in 100 runs and in 10 sets of 10 runs each for the datasets 12-9 and 21-9.

f_0, the higher the f_0, the larger the circle.

This simplification of the objectives enables the representation in a 2D chart, but the charts must be assessed carefully. For example, in Table 5.2 solutions 4 and 5 are similar if we consider f_0, f^l_{total} and f^t_{total}, but if f^l_i and f^t_i are considered, they are no longer similar. f^l_2 is better in solution 4 and f^l_3 is better in solution 5. There are also solutions in some datasets where, if only f_0, f^t_{total} and f^l_{total} are analyzed, then the conclusion is that one solution dominates another one. However, this conclusion is not correct if we consider f^l_i and f^t_i.

For the dataset 21-10 a smooth Pareto front can be clearly identified in the chart. This is partly due to the fact that this dataset produces in almost all runs more than 40 solutions. For the dataset 34-10, there are three solutions aligned with each other - namely (13,7), (14,8) and (15,9) - that are part of the Pareto front even in the simplified representation: the size of the circle is smaller for (14,8) than the one for (13,7) and it is even smaller for (15,9).

5.7.4.1 Datasets 12-9 and 21-9

For the datasets 12-9 and 21-9, only one non-dominated solution was found. Both solutions optimize all f^t_i and all f^l_i. A lower bound for those values can be estimated by considering the number of students in the dataset and the maximum number of students per class. The non-dominated solution reaches this lower bound for both datasets.

Regarding f_0, a lower bound such as the one mentioned above, is not straightforward. Nevertheless, the three approaches can be compared to each other. The minimal values for 12-9, $f_0 = 8$, and for 21-9, $f_0 = 12$, are found by using the three approaches in several runs.

Table 5.3 details the comparison between ACO, HS and NSGA-II. NSGA-II clearly outperforms the other approaches in those datasets, except when runs are grouped in sets of 10 runs each for the dataset 12-9. In that case, the three methods are similar, no single method outperforms the others.

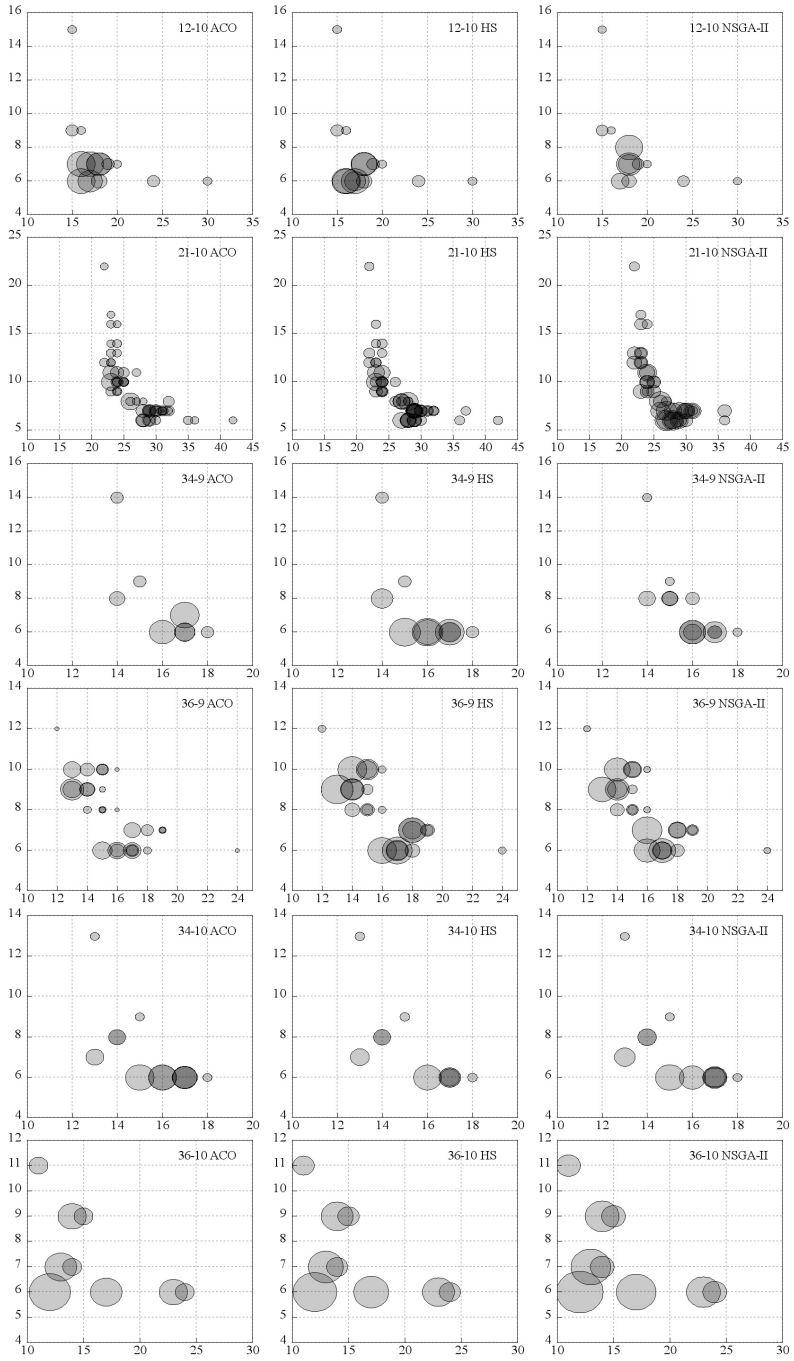

Figure 5.2: Non-dominated solutions for six datasets. Horizontal axis: Sum of f_i^t. Vertical axis: Sum of f_i^l. Size of the circles: f_0. Left: ACO approach. Middle: HS approach. Right: NSGA-II approach.

Dataset	ACO	HS	NSGA-II	Best
36-10	0.2	10.8	2.2	ACO
34-10	2.6	9.4	3.4	ACO
21-10	243	263	313	ACO
12-10	2.8	13.2	3.7	ACO
36-9	0.7	10.6	1.7	ACO
34-9	2.2	8.8	2.9	ACO
12-9	0.8	8.3	1.6	ACO
21-9	9.6	15.6	10.5	ACO

Table 5.4: Comparison of the computation time in seconds for each approach in the six datasets.

5.7.5 Computation time

Table 5.4 contains the average computation time for one run. Those values are calculated as the average of the 100 runs. ACO is always faster than the other approaches. The dataset 21-10 is the biggest dataset, with the highest number of students and of classes. The computation time for this dataset is also the longest.

5.7.6 Hypervolume Comparison

Table 5.5 contains the portion of the hypercube dominated by each approach in the six datasets with the simple and the smoothing hypervolume methods. NSGA-II outperforms ACO and HS in one dataset, while ACO dominates in two datasets. In the three other datasets, there is no significant difference between the three methods, all of them are competitive. The portion of the hypercube covered by the SHC is on average 11% greater than by HC.

5.8 Conclusions

In this chapter, we have presented three approaches applied to a many-objective optimization problem, based on Ant Colony Optimization, Harmony Search and Genetic Algorithms. All approaches have been applied to eight real datasets coming from schools in Switzerland, where the allocation of students to classes must be done by optimizing the pedagogical objective and minimizing both the number of lessons and the number of teachers for different courses.

We have used two different methods in order to compare the three approaches, both of them based on the hypervolume indicator. Our results show that the NSGA-II approach outperforms the other approaches in four datasets out of eight. The ACO approach outperforms the others in one dataset. For the other three datasets, the three approaches are competitive and the three of them provide good solutions.

Regarding the computation time, the ACO approach outperfoms the others. It is important

Simple Hypervolume, 100 runs				
Dataset	ACO	HS	NSGA-II	Best
36-10	61% ± 6.8%	70.0% ± 3.8%	75.0% ± 0.6%	NSGA-II
34-10	74% ± 0.5%	70.4% ± 0.7%	70.3% ± 1.5%	ACO
21-10	76% ± 1.2%	76.2% ± 1.0%	80.0% ± 1.1%	NSGA-II
12-10	81% ± 1.4%	82.0% ± 2.4%	81.0% ± 1.3%	-
36-9	50% ± 0.8%	52.0% ± 0.9%	51.0% ± 1.2%	-
34-9	61% ± 1.3%	62.0% ± 2.0%	62.0% ± 1.3%	-
Smoothing Hypervolume, 10 sets of 10 runs				
Dataset	ACO	HS	NSGA-II	Best
36-10	52% ± 3.2%	57.0% ± 4.9%	65.0% ± 8.5%	NSGA-II
34-10	70% ± 1.3%	64.0% ± 2.0%	63.0% ± 6.7%	ACO
21-10	69% ± 2.9%	72.0% ± 1.3%	75.0% ± 1.9%	NSGA-II
12-10	73% ± 2.2%	77.0% ± 2.3%	76.0% ± 3.0%	HS/NSGA-II
36-9	45% ± 1.1%	47.0% ± 1.6%	46.0% ± 3.8%	-
34-9	53% ± 3.5%	54.0% ± 2.5%	55.0% ± 1.9%	-

Table 5.5: Comparison of the portion of the hypercube dominated by each approach in the six datasets with the simple hypervolume and the smoothing hypervolume.

to remark that the computation is much higher for the biggest dataset. The number of available classes has an important impact on this time since the number of possible grouping of classes increases exponentially with the number of classes.

Chapter 6

Hybrid Metaheuristics to Manage Complex Vehicle Routing Problems

This chapter describes and tackles a complex Vehicle Routing Problem with many different constraints: time windows, heterogeneous fleet, multiple depots, multiple routes and incompatibilities. Five different approaches are presented and applied to fifteen real datasets. Those approaches are based on two metaheuristics, Ant Colony Optimization (ACO) and Genetic Algorithm (GA) that are applied in their standard formulation and combined as hybrid metaheuristics. The results are compared with two commercial tools currently used in the company that deals with this VRP.

6.1 Introduction

The problem presented in this chapter is a Vehicle Routing Problem (VRP) with a heterogeneous fleet of capacitated vehicles, multiple depots, multiple trips and time-windows.

As defined in Section 2.3.3, a VRP consists in finding routes to serve a given number of customers from one or several depots. In the first variant of the VRP introduced in [29], a set of customers were delivered by a fleet of similar vehicles from one depot. Since that simple variant, several variants were studied, some of them are presented in Section 2.3.3.

The variant analyzed in this chapter has the following characteristics for the vehicles, the customers and the depots:

- The fleet is heterogeneous regarding:
  - **Capacity**. The capacity of each vehicle might be different from the others.
  - **Availability**. Vehicles have time restrictions and cannot be used during the whole day, either because of driving time or because of availability of drivers.

  - **Features**. Some vehicles have features that are needed by some customers to load the goods. Those vehicles can also serve the customers that don't need those features.

- There are several depots that can be a destination for the goods, a parking lot for the vehicles (initial and end point) or both.
- The customers have the following characteristics:
  - **Time-windows**.
  - **Loading time**. Once the vehicle arrives at the customer, a moment is needed to load the goods on the vehicle.
  - **Features**. Some of the customers need a special feature in the vehicle that serves them.
  - **Incompatibilities**. Some customers cannot be served within the same route, but can be served in different routes by the same vehicle.
  - A customer is assigned to a depot.

This variant has thus several constraints that have to be satisfied in order to have a feasible solution, this variant is named maVRP for multiple attributes VRP. Its objective is to minimize the total cost that is the sum of two components:

- The distance cost is proportional to the number of kilometers of the solution.
- The time cost is proportional to the total duration of all the routes of the solution.

The following section presents the model that is used in the different approaches proposed for the maVRP.

6.2 Problem statement for the maVRP

In the Traveling Salesman Problem (TSP), the goal is to find a sequence of customers so that all customers are part of the route and the cost is minimized, [92]. The model for the maVRP variant presented in this paper is similar to the TSP, a sequence has to be found with a maximum number of customers served so that the total cost is minimized.

Let us consider that m vehicles are available to serve n customers. Let us denote the set of vehicles as $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ and the set of customers as $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. The objective is to find a permutation $X = \{X_1, X_2, \ldots, X_n\}$ of the n customers, i.e. a sequence of positions where each position corresponds to a customer. A vehicle $V(X_i) \in \mathcal{V}$ should be allocated to each position X_i, if a position X_i has no vehicle allocated, then the corresponding customer is not served. As a vehicle may do several routes, when a vehicle is allocated to a position, this position must be assigned to a route in this vehicle. If a vehicle serves k customers, then it has k routes or less, and the routes are numbered sequentially starting in 1.

Figure 6.1 illustrates a set of routes that serve $n = 7$ customers with three vehicles, V_1, V_3 and V_4. The sequence of the customers is $X = \{X_1, X_2, \ldots, X_7\} = \{C_2, C_3, C_4, C_1, C_6, C_7, C_5\}$.

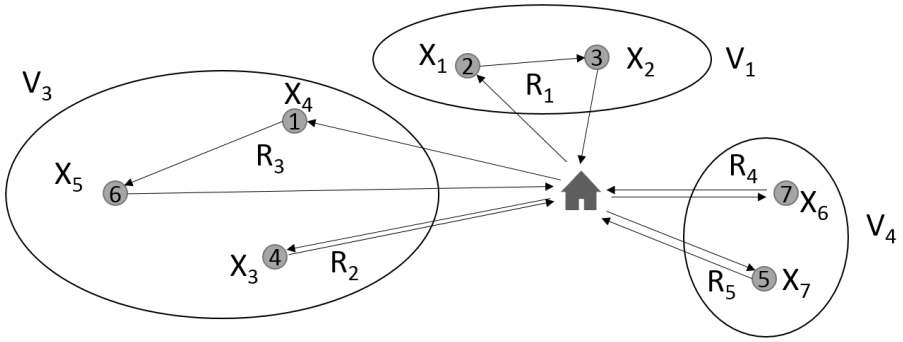

Figure 6.1: Example. $m = 3$ vehicles serve $n = 7$ customers in 5 routes. The sequence of customers is $X = \{C_2, C_3, C_4, C_1, C_6, C_7, C_5\}$

In the sequence, each customer has a successor, for example C_1 (X_4) is the successor of C_4 (X_3). Vehicle V_1 is allocated to the two first positions of the sequence, X_1 and X_2, both positions are allocated to the same route R_1. Vehicle V_3 serves one customer in its first route R_2 and two customers in R_3. Vehicle V_4 serves two customers in two different routes.

Definition 6.2.1. A *solution* $\delta = \{\mu, \mu_V, \mu_R\}$ to the maVRP is defined as the set of three assignments.

- μ is the assignment of a successor to each position in the sequence, $\mu(i) \in \mathcal{C}$ is the successor of X_i, that is $\mu(i) = X_{i+1}$ and $\mu(n) = X_1$, $\forall i \in \{1, \ldots, n\}$.
- μ_V is the assignment of one vehicle to each position. If a vehicle is allocated to X_i, then $\mu_V(i) \in \mathcal{V}$, if not, then $\mu_V(i) = NULL$.
- μ_R is the assignment of one route to each position assigned to a vehicle, that is if $\mu_V(i) \in \mathcal{V}$, then $\mu_R(i) \in \{1, \ldots, n\}$, if not, then $\mu_R(i) = NULL$.

Without any other constraint, there might be several identical solutions with different sequences X. For example, with $n = 2$ and $m = 2$, the solution with $X = \{C_1, C_2\}$, $\mu_V(X_1) = V_1$ and $\mu_V(X_2) = V_2$ is identical to the solution with $X = \{C_2, C_1\}$, $\mu_V(X_1) = V_2$ and $\mu_V(X_2) = V_1$. Therefore, the model includes Expressions 6.1 to 6.6 to avoid this situation. Expression 6.2 assumes that the vehicles are numbered according to a criterion, that might be the plate number or the capacity for example; the vehicle allocated to a position has a number greater than or equal to the vehicle allocated to the previous position. Expression 6.1 states that if a position is not allocated to a vehicle, the subsequent positions are not allocated either. Expression 6.3 says that if the first position of the sequence is allocated to a vehicle, then the corresponding route for this position is 1. Expression 6.4 states that if no vehicle is allocated to a position, then no route is allocated either. If two consecutive positions are allocated to different vehicles, Expression 6.5 implies that the routes allocated to those positions are consecutive. If two consecutive positions are allocated to the same vehicle, Expression 6.6 states that the routes of those positions are the same or consecutive.

$$\forall i \in \{1,\dots,n-1\} : \mu_V(i) \notin \mathcal{V} \Rightarrow \mu_V(\mu(i)) \notin \mathcal{V} \quad (6.1)$$

$$\forall i \in \{1,\dots,n-1\} : \mu_V(i) = V_k \in \mathcal{V} \Rightarrow \begin{cases} \mu_V(\mu(i)) = V_t \in \mathcal{V} & \text{where } t \geq \text{k, or} \\ \mu_V(\mu(i)) \notin \mathcal{V} \end{cases} \quad (6.2)$$

$$\mu_V(1) \in \mathcal{V} \Rightarrow \mu_R(1) = 1 \quad (6.3)$$

$$\forall i \in \{1,\dots,n\} : \mu_V(i) \notin \mathcal{V} \Rightarrow \mu_R(i) = NULL \quad (6.4)$$

$$\forall i \in \{1,\dots,n-1\} : \mu_V(i) \neq \mu_V(\mu(i)) \in \mathcal{V} \Rightarrow \mu_R(\mu(i)) = \mu_R(i) + 1 \quad (6.5)$$

$$\forall i \in \{1,\dots,n-1\} : \mu_V(i) = \mu_V(\mu(i)) \in \mathcal{V} \Rightarrow \mu_R(\mu(i)) = \begin{cases} \mu_R(i) & \text{or} \\ \mu_R(i) + 1 \end{cases} \quad (6.6)$$

The order of the customers in the sequence determines also the order in which those customers are served. If X_i and X_{i+k} are assigned to the same vehicle, X_i is served before X_{i+k}, $\forall i, k \in \{1,\dots,n-1\}$. It is important to note that even if X_{i+1} is the successor of X_i in the sequence of X, this doesn't mean that both of them are allocated to the same route or to the same vehicle. Indeed, even if X_i is allocated to a vehicle, X_{i+1} might be unallocated.

Definition 6.2.2 defines a consistent solution, that is a solution that is feasible for the maVRP. The Condition 4 in this definition includes all constraints that are specific to the addressed maVRP, for example, the ones related to the time windows of the customers or to the capacity of the vehicle. An exhaustive list is not possible since there are many of those constraints, but a few examples are given in Section 6.1 and more can be found in [20].

Definition 6.2.2. A solution $\delta = \{\mu, \mu_V, \mu_R\}$ to the maVRP is *consistent* if:

1. $\mu(i) \in \{1,\dots,n\}, \forall i \in \{1,\dots,n\}$. All customers have a successor.
2. $\mu(i) \neq \mu(j), \forall i \neq j \in \{1,\dots,n\}$. A customer cannot be the successor of two different customers.
3. Expressions 6.1 to 6.6 are satisfied.
4. All constraints of the maVRP related to the vehicle and route assignments are satisfied: time, capacity and compatibility constraints.

A consistent solution may not allocate all customers to a vehicle. If all customers are allocated, the solution is said to be complete as defined in Definition 6.2.3.

Definition 6.2.3. A solution $\mathcal{S} = \{\mu, \mu_V, \mu_R\}$ to the maVRP is *complete* if:

1. It is consistent
2. $\mu_V(i) \in \mathcal{V}$ and $\mu_R(i) \in \{1,\dots,n\}, \forall i \in \{1,\dots,n\}$ (i.e.all customers are assigned to a vehicle and to a route).

Definition 6.2.4 gives the criteria to compare the quality of two solutions.

Definition 6.2.4. A solution $\delta = \{\mu, \mu_V, \mu_R\}$ is said to be *better* than a solution $\delta' = \{\mu', \mu'_V, \mu'_R\}$ if at least one of the two following conditions is met:

- More customers are allocated to a vehicle in δ than in δ':
  $Card(\{i \in \{1, \ldots, n\} : \mu_V(i) \in \mathcal{V}\}) > Card(\{i \in \{1, \ldots, n\} : \mu'_V(i) \in \mathcal{V}\})$,
  where $Card(\cdot)$ denotes cardinality of a set.
- The same number of customers are assigned to a vehicle in δ and in δ', but the total cost of δ is smaller than the total cost of δ'.

6.3 The ACO approach

Algorithm 17 describes the steps of the ACO approach. Three solutions are used to store the current solutions:

- δ_{ant} is the solution found by the current ant.
- δ_{best} is the best solution found in the current iteration.
- $\delta_{Gl.best}$ is the best solution found in all the previous iterations.

In line 7, *Succ* is the set of the possible successors for a customer. This set is updated when a successor is assigned to a customer, line 12, since in the maVRP model described in Section 6.2 a customer can be the successor of only one customer.

In line 8, $p_f(k)$ is the probability that the customer C_k is assigned to the first position of the sequence, i.e. the probability that $\mu(n) = k$. This assignment is also performed by the ant, but the pheromones associated with it are independent from the pheromones associated with the successors.

In line 11, $p(i, k)$ is the probability that customer C_k becomes the successor of C_i, i.e. the probability that $\mu(i) = k$. This probability is calculated using the pheromones associated with the assignment of successors.

We have thus two sets of pheromones:

- τ_{ij} is related to the assignment of the customer j as the successor of the customer i in the sequence.
- τ_k is related to the assignment of the customer k to the first position of the sequence.

Both pheromones are updated at the end of an iteration. Even if the first position selection cannot be viewed as an ACO approach, the use of pheromones enables to favor the selection of customers who may lead to good solutions when they are set to the first position of the sequence.

Algorithm 17: ACO algorithm for the maVRP

Input: Parameters of the ACO approach
Output: Solution to the maVRP

```
1  Initialize pheromones
2  δ_Gl.best = ∅
3  for t ← 1, ..., N do
4      δ_best = ∅
5      for ant ← 1, ..., N_ants do
6          μ_ant = ∅
7          Succ = {1, 2, ··· , n}
8          Select f ∈ Succ with a probability p_first(f) → μ(n) = k
9          Succ ← Succ\{k}
10         for i ← 1, ..., n do
11             Select k ∈ Succ with probability p(i, k) → μ(i) = k
12             Succ ← Succ\{k}
13         Assign vehicles and routes → δ_ant
14         Compute the cost and the number of customers with no vehicle assigned for δ_ant
15         if δ_ant is better than δ_best then
16             δ_best ← δ_ant
17     Update pheromones and probabilities
18     if δ_best is better than δ_Gl.best then
19         δ_Gl.best ← δ_best
```

In line 15, δ_{best} is replaced by δ_{ant} only if this one is better as described in definition 6.2.4.

The assignment of a vehicle and a route to a customer, line 13, is described in algorithm 18. The first node to be served, f, is X_1 in the maVRP model and is the first position to be assigned to a vehicle. This algorithm guarantees that Expressions 6.1 to 6.6 are satisfied since a customer is assigned to a vehicle only if the predecessor has already been assigned. In line 3, the current vehicle $CurrVehicle$, the current route $CurrRoute$ and the current position of the sequence $CurrPosition$ are initialized. The allocation always starts with the first vehicle V_1 as mentioned previously and the first customer to which a successor is assigned is the first customer of the sequence f.

In line 6, the customer in the current position is assigned to the current vehicle and route if this assignment produces a consistent solution, that means that μ has to satisfy conditions 1 and 3 from Definition 4. If not, it is assigned to a new route in line 12, but only if this assignment produces a consistent solution. If not, the next vehicle is tested for this position. With this strategy the allocation of vehicles to the positions of the sequence stops as soon as the solution is no longer consistent if a vehicle is allocated to the next position.

6.3.1 The pheromones' update

The parameters in the ACO algorithm are critical regarding the convergence of the search and the diversification of the explored solutions. In the Max-Min Ant System (MMAS), [141], the

Algorithm 18: Assignment of vehicles and routes

Input: Assignment of the successors μ, First node of the sequence f
Output: Assignment of a vehicle and a route to each position

```
for i ← 1, ..., n do
    μ_V(i) = NULL; μ_R(i) = NULL
CurrRoute = 1; CurrVehicle = 1; CurrPosition = f
while CurrVehicle ≤ m do
    if CurrPosition can be assigned to CurrVehicle and to CurrRoute then
        μ_V(CurrPosition) = CurrVehicle
        μ_R(CurrPosition) = CurrRoute
        CurrPosition = μ(CurrPosition)
    else
        CurrRoute = CurrRoute + 1
        if CurrPosition can be assigned to CurrVehicle and to CurrRoute then
            μ_V(CurrPosition) = CurrVehicle
            μ_R(CurrPosition) = CurrRoute
            CurrPosition = μ(CurrPosition)
        else
            CurrVehicle = CurrVehicle + 1
```

value of the pheromones are bounded in order to avoid stagnation on a specific solution or around a solution that would be mainly due to a situation where the quantity of pheromones deposited on the assignments of this solution is much higher than the quantity of pheromones deposited on other assignments. Another option proposed in the MMAS is the reinitialization of pheromones in order to avoid the stagnation of the search.

The method presented in this section in order to avoid stagnation consists in having bounds whose value is changed dynamically. The initial limits are set to τ_{min} and τ_{max}. If $\delta_{Gl.best}$ has not been updated for a given number of iterations N_{noImp}, the values of those limits are then set to new values τ'_{min} and τ'_{max}, where $\tau_{min} \leq \tau'_{min} \leq \tau'_{max} \leq \tau_{max}$.

Algorithm 19 describes the steps of the pheromones' update. The bounds, UL and LL, used to limit the value of the pheromones are different if the number of iterations without improvement to the global solution has exceeded a predefined limit N_{noImp}. When the pheromones are updated, those bounds are used as upper or lower limits for the value of the pheromones.

6.3.2 The ACO-3-opt approach

ACO is often much more efficient if it is combined with a local search, [52, 88]. In this section, a local search is proposed in order to improve the performance of the simple ACO approach.

A Neighborhood Search (NS) consists in changing a solution iteratively with a small modification each time so that the new solution is close to the previous one. In the ACO approach presented in this section, a NS is added in order to improve the solutions found by the ants. The proposed NS consists in trying to move each customer from its current position to the different

Algorithm 19: Pheromones' update for the ACO approach

Input: Assignment of the successors μ, Current number of consecutive iterations with no improvement n_{noImp}, Increment value Δ

Output: Updated pheromones

```
if n_noImp > N_noImp then
    UL = τ'_max
    LL = τ'_min
    n_noImp = 0
else
    UL = τ_max
    LL = τ_min
for i ← 1, ..., n do
    for i ← 1, ..., n do
        τ(i, j) = max(ρ · τ(i, j); LL)
for i ← 1, ..., n do
    τ(i, μ(i)) = min(τ(i, μ(i)) + Δ; UL)
```

Sequence	C_3	C_5	C_6	C_1	C_4	$\boldsymbol{C_2}$	C_7
New sequence	C_3	$\boldsymbol{C_2}$	C_5	C_6	C_1	C_4	C_7

Table 6.1: Example of the 3-opt local search. C_2 becomes the successor of C_3.

possible positions in the sequence. Figure 6.2 represents an example of such a modification: one customer has been inserted in another position in the sequence. Table 6.1 also contains an example where the customer C_2 becomes the successor of C_3 with such a local modification. This method is similar to the 3-opt local search proposed by Lin in [92] where the TSP is solved step by step replacing 3 arcs by 3 other arcs in a solution.

Algorithm 20 contains the steps of this local search. For all possible pair of customers, one of them becomes the successor of the other, the other successors are not changed.

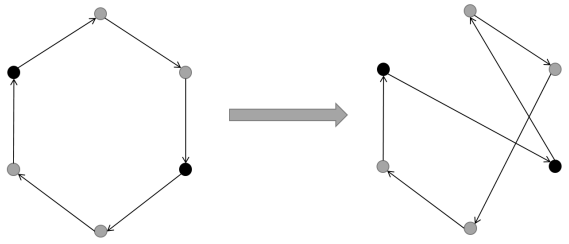

Figure 6.2: Left: Solution μ before local search. Right: Solution μ after a step of the 3-opt local search.

Algorithm 20: Local Search for ACO: 3-opt

Input: Assignment of the successors μ
Output: New assignment μ at least as good as the initial μ

```
1  for i ← 1, ..., n do
2      for j ← 1, ..., n do
3          μ_old ← μ
4          μ(μ^{-1}(j)) = μ(j)
5          μ(j) = μ(i)
6          μ(i) = j
7          Assign vehicles and routes to μ
8          Compute the cost K for μ
9          Compute the number of unassigned
             NU = Card({i ∈ {1, ..., n} : μ_V(i) = NULL})
10         if μ_old is better than μ then
11             μ ← μ_old
```

6.4 The GA approach

As explained in Section 2.1.2, genetic algorithms are based on the natural selection process. They start with an initial population of P_{size} solutions. Those solutions are combined together in order to create a new population, the offspring population. Both populations compete then to be part of the next parent population that includes the best solutions of both of them.

In the GA approach presented in this section, the mutation process is applied in a different way than in the classic GA. Two parents are always selected and combined in order to generate an offspring solution. The mutation is then applied to this new solution with a probability $Prob_{mutation}$.

Algorithm 21 describes the steps of the GA approach. The population is first filled with random solutions, line 2. At each iteration and for each parent in the population, another parent is selected with a probability $p(j)$, line 11. This probability depends on the quality of the solution, it is higher for better solutions.

Once the second parent is selected, for each customer, a successor is selected. This successor is the same as in the first parent or as in the second parent with a probability of 0.5 each, as long as the successor has not been already assigned as successor to another customer, lines 13 to 16. If it has been previously assigned, the current customer is set aside and a successor is randomly assigned once the crossover is finished, line 17.

The mutation operation, line 18, is the same as the local search 3-opt used in the ACO approach and is described in algorithm 20. It consists in trying to move each customer from its current position to the different possible positions in the sequence. Figure 6.3 represents an example of a mutation: one customer has been inserted in another position in the sequence.

The assignment of vehicles and routes, line 19, is done as in the ACO approach with algorithm 18. The solution δ_{best}, line 23, is used to store the best solution found so far by the algorithm. At the end of the iteration, only the P_{size} best solutions among parent and offspring solutions

Figure 6.3: Left: Solution μ before mutation. Right: Solution μ after a mutation.

are selected to compose the next parent population, line 24. The selection of the first customer in the sequence is done as in the ACO approach, line 7. The pheromones and the probabilities are updated at the end of the iteration, line 25.

This process is repeated for several iterations and at the end of the last iteration, the population contains the best solutions found.

6.5 The ACO-GA approach

As mentioned in Section 2.1.1.4, ACO is more efficient when combined with a local search. In Section 6.3.2, a first local search method is proposed. In this section, another local search based on GA is proposed. The new combined metaheuristic is named ACO-GA.

In addition to the pheromones, the ACO-GA approach stores a population of solutions that contains the best solutions found so far. Once an ant has built a solution, this solution is combined with the solutions of the population in the same way as two parents are combined in the basic GA approach presented in Section 6.4.

Algorithm 22 describes the steps of this approach. In line 7, the ant finds a solution as in algorithm 17 from line 5 to line 14. In line 10, the solution found by the ant is combined with the solutions of the population as in the GA approach in algorithm 21, from line 12 to line 19. If the combined solution is better than the original ant solution, this one is replaced immediately by this new solution. After the combination, if the solution found by the ant is better than the worst solution stored in the population P, the latter is replace by the ant's solution.

As in the original ACO approach, δ_{best} and $\delta_{Gl.best}$ are used to store the best solution of the current iteration and the global best solution. At the end of the iteration, the pheromones are updated for both the sequence and the first customer in the sequence, and the new probabilities are calculated for the next iteration as in algorithm 17.

6.6 The GA-ACO approach

The last proposed approach is also a combination of GA and ACO, the GA-ACO, but GA being this time the main metaheuristic.

The GA-ACO algorithm is very similar to the original GA presented in Section 6.4. The only difference lies in the choice of the successor for the customers that were not assigned by

Algorithm 21: GA for the maVRP.

Input: Parameters for the GA approach
Output: Population of solutions to the maVRP

```
1  δ_best = ∅
2  Populate P_0 with P_size random solutions
3  for t ← 1, ..., N do
4      R_t = P_{t-1}
5      for μ_i ∈ P_{t-1} do
6          Succ = {1, 2, ··· , n}
7          Select k ∈ Succ with a probability p_f(k) → μ(n) = k
8          Succ ← Succ\{k}
9          for i ← 1, ..., n do
10             μ(i) = NULL
11         Select the second parent μ_j ∈ P_{t-1}, j ≠ i with a probability p(j)
12         for i ← 1, ..., n do
13             Select r ∈ {i, j} with a probability of 0.5 each
14             if μ_r(i) ∈ Succ then
15                 μ(i) = μ_r(i)
16                 Succ ← Succ\{μ(i)}
17         Assign successors to the unserved customers keeping μ consistent
18         Mutate μ with a probability Prob_mutation
19         Assign the vehicles and the routes to μ → δ = {μ, μ_V, μ_R}
20         Compute the cost and the number of customers with no vehicle assigned for δ
21         R_t ← R_t ∪ δ
22         if δ is better than δ_best then
23             δ_best ← δ
24     Select the new parent population P_t in R_t
25     Update the pheromones for the first customer of the sequence and calculate p_first
```

any of the parents, line 17. This selection is random in algorithm 21. In the GA-ACO approach, pheromones are updated at the end of each iteration with the global best solution found so far. Those pheromones are then used to select the successors for the customers that have been set aside. The candidate that has more pheromone has a higher probability to be selected as a successor for a customer. The rest of the GA-ACO algorithm is identical to the initially proposed algorithm 21.

6.7 Experimentation

6.7.1 The datasets

Fifteen datasets have been used to compare the performance of the three approaches. Those datasets come from a company that collects goods at different customers, those customers differ from one day to the other. This company has a fleet of vehicles that can do one or more routes

Algorithm 22: ACO-GA algorithm for the maVRP

Input: Parameters of the ACO approach, Size of the population P_{size}
Output: Solution to the maVRP

```
1  Initialize pheromones
2  Initialize population P = {δ_1, ..., δ_{P_size}}
3  δ_{Gl.best} = ∅
4  for t ← 1, ..., N do
5      δ_best = ∅
6      for ant ← 1, ..., N_ants do
7          Find a solution δ_ant
8          δ_{Ant.best} = δ_ant
9          for i ← 1, ..., P_size do
10             Combine δ_ant with δ_i → δ_C
11             if δ_C is better than δ_{Ant.best} then
12                 δ_{Ant.best} ← δ_C
13         if δ_{Ant.best} is better than δ_best then
14             δ_best ← δ_{Ant.best}
15         Update P with δ_{Ant.best}
16     Update pheromones and probabilities
17     if δ_best is better than δ_{Gl.best} then
18         δ_{Gl.best} ← δ_best
```

per day and a set of depots where goods have to be delivered.

Table 6.2 contains some information about the datasets used in this chapter. For each dataset, it gives the number of depots, customers and vehicles. It contains also the average weight that has to be collected at the customers and the average capacity of the vehicles.

Each dataset has three or four types of goods which cannot be loaded within the same route, but can be loaded in two different routes of the same vehicle. The use, working and driving times depend on the vehicle and vary between 4h00 and 18h00. The loading and unloading times vary from 20 to 90 minutes, and depend on the customer or on the depot respectively.

6.7.1.1 The hard constraints

As the proposed model and the approaches are adapted to any type of maVRP, the specific constraints of the problem used for the experimentation are described in this section. Before that, the different times of a route from its start time to its end time are listed here:

- **Driving times**: from the parking lot of the vehicle to the first customer, between customers; from the last customer to the destination depot of the loaded goods; from the destination depot to the parking lot of the vehicle (only for the last route of the vehicle).
- **Waiting times**: to load the goods at the customer's facility; if the arrival time at a customer's location is before its time window; to unload the goods at the depot.

Dataset	Depots	Customers	Aver. Weight	Vehicles	Aver. Cap.
07-13	5	42	4653	12	18083
07-14	6	53	3812	12	18083
07-15	6	56	3689	12	18083
07-18	6	43	4561	12	18083
07-19	6	43	4561	12	18083
10-11	5	44	4309	11	18364
10-13	5	43	4100	11	18364
10-14	5	56	3439	12	18083
10-17	5	37	4308	10	18000
10-18	5	45	4200	10	18700
10-19	6	47	3928	11	18364
10-20	5	44	3805	9	19111
10-21	5	54	3554	11	18364
10-24	5	38	4479	10	18700
10-25	5	48	3958	11	18364

Table 6.2: Datasets. Number of depots, customers and vehicles per dataset. Average weight of the customers' goods and average capacity of the vehicles.

The constraints that have to be satisfied are the following:

- **Time constraints**: the loading time of a customer is included in its time window; the driving time of a vehicle does not exceed its maximum driving time; the use time of a vehicle (from the start time of the first route to the end time of the last route) does not exceed its maximum use time; the working time of a vehicle (sum of the driving and waiting times) does not exceed its maximum working time.
- **Capacity constraints**: the capacity of a vehicle is not exceeded.
- **Compatibility constraints**: the features of the vehicle allocated to a customer are adapted to this customer's requirements; all goods loaded simultaneously in a vehicle are compatible and have the same destination depot; a vehicle starts and finishes the day at its depot, but can deliver goods to any depot.

6.7.1.2 The objective

The main objective is to serve as many customers as possible. The second objective is to minimize the total cost which includes the distance cost and the time cost. The distance cost is proportional to the total number of kilometers. The time cost is proportional to the total working hours.

ACO			
ρ	0.98	φ	3
τ_{init}	15		
τ_{min}	1	τ_{max}	30
τ'_{min}	5	τ'_{max}	15

Table 6.3: Experimentation. Parameters for the ACO approaches.

Those objectives cannot be combined together. As explained in definition 6.2.4 a solution is better than another if more customers are served, no matter the cost of the solutions.

6.7.2 The parameters

The five approaches have been applied to the fifteen datasets. Two commercial tools currently used by the company have also been used in order to compare the results of the three approaches. The first tool is based on the savings algorithm, the second tool is based on a branch-and-bound algorithm.

For the ACO approach, the number of iterations was set to 2000, for the ACO-3-opt and ACO-GA, it was set to 400, for the GA and the GA-ACO, it was set to 800. The population or colony size was 50 individuals. Ten runs were launched for the five approaches. The parameters for ACO are given in table 6.3. The mutation parameter used for the GA approach is 0.1. The population size for the local search GA in the ACO-GA approach is 5 individuals. The value of those parameters were set experimentally.

6.7.3 The results

The results of the experimentation with the fifteen datasets are presented in this section. The first part, Section 6.7.3.1, compares the three different ACO approaches, the simple ACO, and the two ACO with local search ACO-3-opt and ACO-GA. The second part, Section 6.7.3.2, contains the same comparison for the two GA approaches, the simple GA and the combined GA-ACO.

The last part, Section 6.7.3.3, is dedicated to the comparison of the five approaches and the two commercial tools. The comparison is done regarding the two objectives, to serve all customers and to minimize cost, and the computation time.

6.7.3.1 Results for the ACO approaches

Table 6.4 contains the results for the three ACO approaches. For the simple ACO approach, without local search, two data are given: the average number of customers that are not served, i.e. not assigned to a vehicle, and the average time for one run. For the two other approaches, with 3-opt and with GA for the local search, three data are given: the number of complete

Dataset	ACO		ACO-3-opt			ACO-GA		
	Unass.	Time	Comp.	Cost	Time	Comp.	Cost	**Time**
07-13	13 ± 2	00:40	8	**3609 ± 9**	04:47	**9**	3631 ± 16	02:21
07-14	24 ± 2	01:37	5	3944 ± 62	06:55	**8**	**3936 ± 24**	04:09
07-15	28 ± 1	01:58	1	**3902 ± 0**	09:04	**4**	3948 ± 37	03:46
07-18	16 ± 1	01:06	1	**3787 ± 0**	04:51	**5**	3964 ± 91	02:42
07-19	16 ± 2	01:05	4	**3818 ± 34**	04:47	**8**	3884 ± 87	02:41
10-11	17 ± 2	01:05	1	**3642 ± 0**	05:06	**8**	3704 ± 84	02:50
10-13	15 ± 2	01:06	2	**3143 ± 0**	04:52	**7**	3144 ± 1	02:41
10-14	21 ± 4	01:29	8	**3453 ± 83**	08:29	**10**	3579 ± 75	04:50
10-17	11 ± 3	00:39	5	**2872 ± 3**	03:10	**7**	2915 ± 92	01:41
10-18	17 ± 2	01:03	0	-	-	0	-	-
10-19	16 ± 2	01:06	**10**	3356 ± 74	05:38	8	**3293 ± 82**	03:36
10-20	18 ± 2	01:05	3	**3129 ± 13**	04:16	**8**	3163 ± 36	02:47
10-21	19 ± 2	01:15	1	**3549 ± 0**	08:49	**4**	3758 ± 46	04:20
10-24	10 ± 2	00:44	7	**2919 ± 2**	03:45	**9**	2922 ± 4	01:59
10-25	17 ± 4	01:07	**7**	**3565 ± 59**	06:31	5	3691 ± 87	03:12

Table 6.4: Experimentation. Results for the ACO approaches.

solutions, that is with all customers served, out of the ten runs, the average cost and the average time for one run.

Even if the simple ACO without local search is much faster, the results are the worst, it doesn't find any solution with all customers served in any dataset. The ACO-3-opt finds solutions with a lower cost, but the number of complete solutions found is almost always lower and the computation time is always higher compared to ACO-GA.

6.7.3.2 Results for the GA approaches

Table 6.5 contains the results for the two GA approaches. Regarding the number of complete solutions, there is no significant difference, GA is better in 5 datasets, GA-ACO is better in 6 of them. Regarding the cost, when both methods find complete solutions, the difference is not significant, except for the datasets 07-19 and 10-25, where GA outperforms GA-ACO.

The fact that GA-ACO is not outperforming GA might be explained by the fact that the randomness in the GA approach introduces more diversity than in the GA-ACO. In GA-ACO, the selection pressure is higher since assignments that produced better results previously are selected with a higher probability.

Dataset	GA			GA-ACO		
	Complete	Cost	Time	Complete	Cost	Time
07-13	**5**	3626 ± 14	00:44	1	**3603 ± 0**	00:49
07-14	2	**3914 ± 5**	01:26	**3**	3933 ± 29	01:29
07-15	**1**	3854 ± 0	01:07	**1**	**3851 ± 0**	01:09
07-18	**2**	**3798 ± 12**	00:53	**2**	3807 ± 21	00:59
07-19	2	**3809 ± 23**	00:52	**3**	3994 ± 61	01:00
10-11	**2**	**3642 ± 0**	00:55	0	-	-
10-13	**1**	**3143 ± 0**	00:54	0	-	-
10-14	**4**	**3522 ± 63**	01:28	1	3534 ± 0	01:47
10-17	**1**	2875 ± 0	00:37	**1**	**2870 ± 0**	00:39
10-18	0	-	-	0	-	-
10-19	**9**	3290 ± 100	00:58	7	**3276 ± 125**	01:14
10-20	3	**3120 ± 0**	01:02	**4**	**3120 ± 0**	01:11
10-21	0	-	-	**1**	**3548 ± 0**	01:15
10-24	1	**2916 ± 0**	00:44	**2**	2918 ± 3	00:46
10-25	6	**3653 ± 39**	01:12	**7**	3699 ± 138	01:16

Table 6.5: Experimentation. Results for the GA approaches.

6.7.3.3 Comparison

First objective: Complete solutions

Table 6.6 compares the success of the tools and four approaches in finding a complete solution, that is with all customers served, which is the first objective of the problem. The simple ACO is not competitive and therefore not included in this comparison.

Tool1 corresponds to the commercial savings approach; it gives always one single solution per dataset and in eight datasets, it does not find a solution with all customers served. When no complete solution is found, the number of customers not served is given into brackets.

Tool2 corresponds to the branch-and-bound approach; it gives always one single solution per dataset and finds always a solution with all customers served, if it exists, no matter the time needed to find it. As there is no time limit for this method to find a complete solution, the comparison of these results is strongly biased.

Regarding the four approaches proposed in this chapter, the ACO-GA clearly outperforms the others, except in two datasets where the results are nevertheless competitive.

None of the proposed approaches, nor Tool1, finds a complete solution for the dataset 10-18 in the 10 runs. This instance is the biggest if we compare the quantity that has to be collected to the capacity of the vehicles. In the other instances, this ratio is between 87% and 97%, this

Dataset	Tool1 Compl.	Tool2 Compl.	ACO-3-opt Compl.	ACO-GA Compl.	GA Compl.	GA-ACO Compl.
07-13	✓	✓	8	**9**	5	1
07-14	- (3)	✓	5	**8**	2	3
07-15	- (7)	✓	1	**4**	1	1
07-18	- (1)	✓	1	**5**	2	2
07-19	- (2)	✓	4	**8**	2	3
10-11	- (2)	✓	1	**8**	2	-
10-13	✓	✓	2	**7**	1	-
10-14	✓	✓	8	**10**	4	1
10-17	✓	✓	5	**7**	1	1
10-18	- (6)	✓	-	-	-	-
10-19	- (4)	✓	**10**	8	9	7
10-20	- (1)	✓	3	**8**	3	4
10-21	✓	✓	1	**4**	-	1
10-24	✓	✓	7	**9**	1	2
10-25	✓	✓	**7**	5	6	**7**

Table 6.6: Experimentation. Comparison of four approaches and two commercial tools regarding the first objective: To serve a maximum of customers. For Tool1 and Tool2: Number of customers not served. For proposed approaches: Number of runs with all customers served out of the ten runs.

ratio is 101% for the dataset 10-18. As several routes are performed by the vehicles, the ratio above 100% is not an issue, but this makes this instance more difficult than the others. Tool2 finds a solution to it, but as mentionned previously, this tool has no time limit for that.

Second objective: Cost

Table 6.7 compares the effectiveness of the tools and the four approaches regarding the cost, which is the second objective of the problem.

For Tool1 and Tool2, as one single solution is found, and the solution is always the same, the data is the cost of this single solution. For the four proposed approaches, if several complete solutions were found, the data given is the average and the standard deviation. If only one complete solution was found, no standard deviation is given. The best results are highlighted in bold, with a margin of 0.1%, that means that a cost which is 0.1% higher than the lowest cost is also considered as a best result and is highlighted, as for dataset 07-15 where GA and GA-ACO are both considered as best.

Both tools and ACO-GA are clearly outperformed by the three other approaches. If the margin is increased to 2%, the only approach that is not competitive is Tool1, the others have between nine and thirteen datasets among the best.

The local search of ACO-3-opt is much more aggressive than the local search of ACO-GA. It

Dataset	Tool1	Tool2	ACO-3-opt	ACO-GA	GA	GA-ACO
07-13	3689	3636	3609 ± 9	3631 ± 16	3626 ± 14	**3603**
07-14	-	3929	3944 ± 62	3936 ± 24	**3914 ± 5**	3933 ± 29
07-15	-	3856	3902	3948 ± 37	**3854**	**3851**
07-18	-	3978	**3787**	3964 ± 91	3798 ± 12	3807 ± 21
07-19	-	3941	3818 ± 34	3884 ± 87	**3809 ± 23**	3994 ± 61
10-11	-	3670	**3642**	3704 ± 84	**3642 ± 0**	-
10-13	3330	3173	**3143 ± 0**	**3144 ± 1**	**3143**	-
10-14	3591	**3423**	3453 ± 83	3579 ± 75	3522 ± 63	3534
10-17	3075	2947	**2872 ± 3**	2915 ± 92	2875	**2870**
10-18	-	**3874**	-	-	-	-
10-19	-	**3249**	3356 ± 74	3293 ± 82	3290 ± 100	3276 ± 125
10-20	-	3301	3129 ± 13	3163 ± 36	**3120 ± 0**	**3120 ± 0**
10-21	3787	3569	**3549**	3758 ± 46	-	**3548**
10-24	3154	2943	2919 ± 2	2922 ± 4	**2916**	**2918 ± 3**
10-25	3775	3621	**3565 ± 59**	3691 ± 87	3653 ± 39	3699 ± 138

Table 6.7: Experimentation. Comparison of the four approaches and the two commercial tools regarding the second objective: To minimize the total cost. For Tool1 and Tool2: Cost of the single solution. For proposed approaches: Average and standard deviation of the cost in the runs when all customers are served out of the ten runs.

tries systematically to move each single customer from its current position to another position in the sequence. This method is more efficient to find a local optima than the local search GA that does not act so systematically since the ant solution is only combined once with each of the five solutions of the GA population.

Computation time

Regarding the computation time, Tool1 finds the solution in two to five seconds on average. For the other approaches, table 6.8 contains the average time needed for one run. No time is given for the datasets and approaches where no complete solution was found. GA and GA-ACO clearly outperform the others.

Evolution of the solutions

Figures 6.4, 6.5, and Appendix B represent the evolution of the number of customers that are not served in the best solution, and in the global best, for the fifteen datasets and the four approaches ACO-3-opt, ACO-GA, GA and GA-ACO. Those results come from one single run with the same seed for all of them. The objective in those charts is to compare the evolution of the quality of the solutions found by the different approaches.

In most of the datasets, the best solution is found after less than half of the iterations. Except for the dataset 07-15, ACO-GA finds the best solution in much less time than ACO-3-opt.

Dataset	Tool2	ACO-3-opt	ACO-GA	**GA**	**GA-ACO**
07-13	05:02	04:47	02:21	00:44	00:49
07-14	03:22	06:55	04:09	01:26	01:29
07-15	04:42	09:04	03:46	01:07	01:09
07-18	04:00	04:51	02:42	00:53	00:59
07-19	04:56	04:47	02:41	00:52	01:00
10-11	03:00	05:06	02:50	00:55	-
10-13	00:44	04:52	02:41	00:54	-
10-14	04:10	08:29	04:50	01:28	01:47
10-17	00:52	03:10	01:41	00:37	00:39
10-18	07:08	-	-	-	-
10-19	05:23	05:38	03:36	00:58	01:14
10-20	01:46	04:16	02:47	01:02	01:11
10-21	13:32	08:49	04:20	-	01:15
10-24	00:55	03:45	01:59	00:44	00:46
10-25	10:50	06:31	03:12	01:12	01:16

Table 6.8: Experimentation. Computation time in mm:ss.

6.8 Conclusions

In this chapter, five different approaches were presented to solve maVRP with many different constraints: time windows, heterogeneous fleet, multiple depots, multiple routes and incompatibilities. Those approaches were applied to fifteen real datasets and the results were compared to two commercial tools currently used in the company that deals with this maVRP.

These approaches are based on well-known metaheuristics, Ant Colony Optimization (ACO) and Genetic Algorithm (GA). Two approaches are hybrid metaheuristics that combine both of them, ACO and GA. In one of them, ACO-GA, ACO is the main metaheuristic and GA is used for the local search. In the other, GA-ACO, GA is the main metaheuristic and ACO is used to break ties instead of using randomness when a decision has to be made in order to complete the solution.

The results of the five approaches have been compared through two different perspectives: quality of the solution and computation time. The quality of the solution has two components, the number of customers that cannot be served and the financial cost of the solution, the first one having a higher priority than the second.

Considering the first objective, maximizing the number of served customers, one of the tools and the ACO-GA approach outperform the others. Considering the second objective, the cost, the GA and GA-ACO provide better results. The computation time needed for one iteration is much better in the GA approaches.

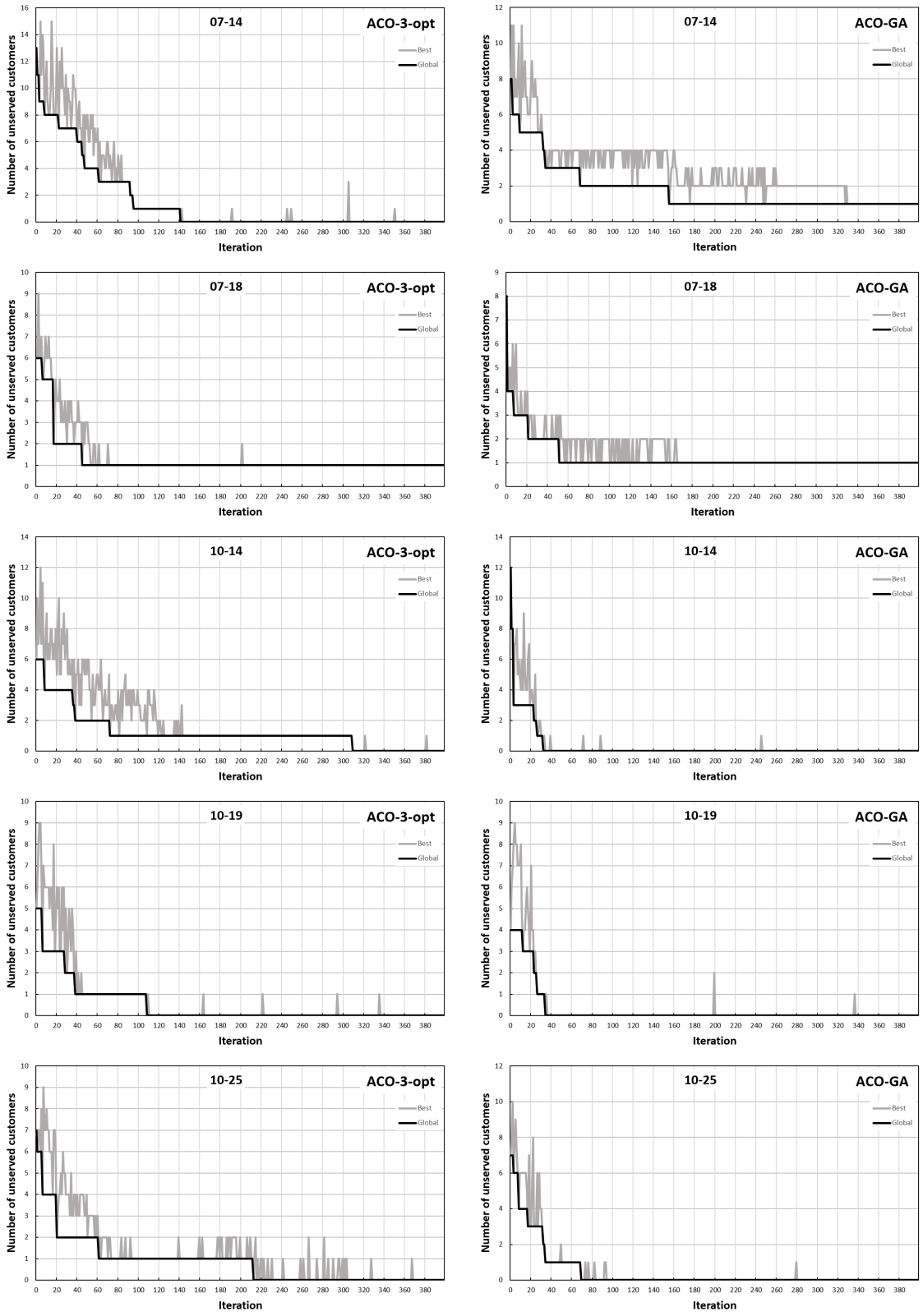

Figure 6.4: Number of customers not served in the best solution and in the global best solution for the **ACO** approaches and 5 datasets.

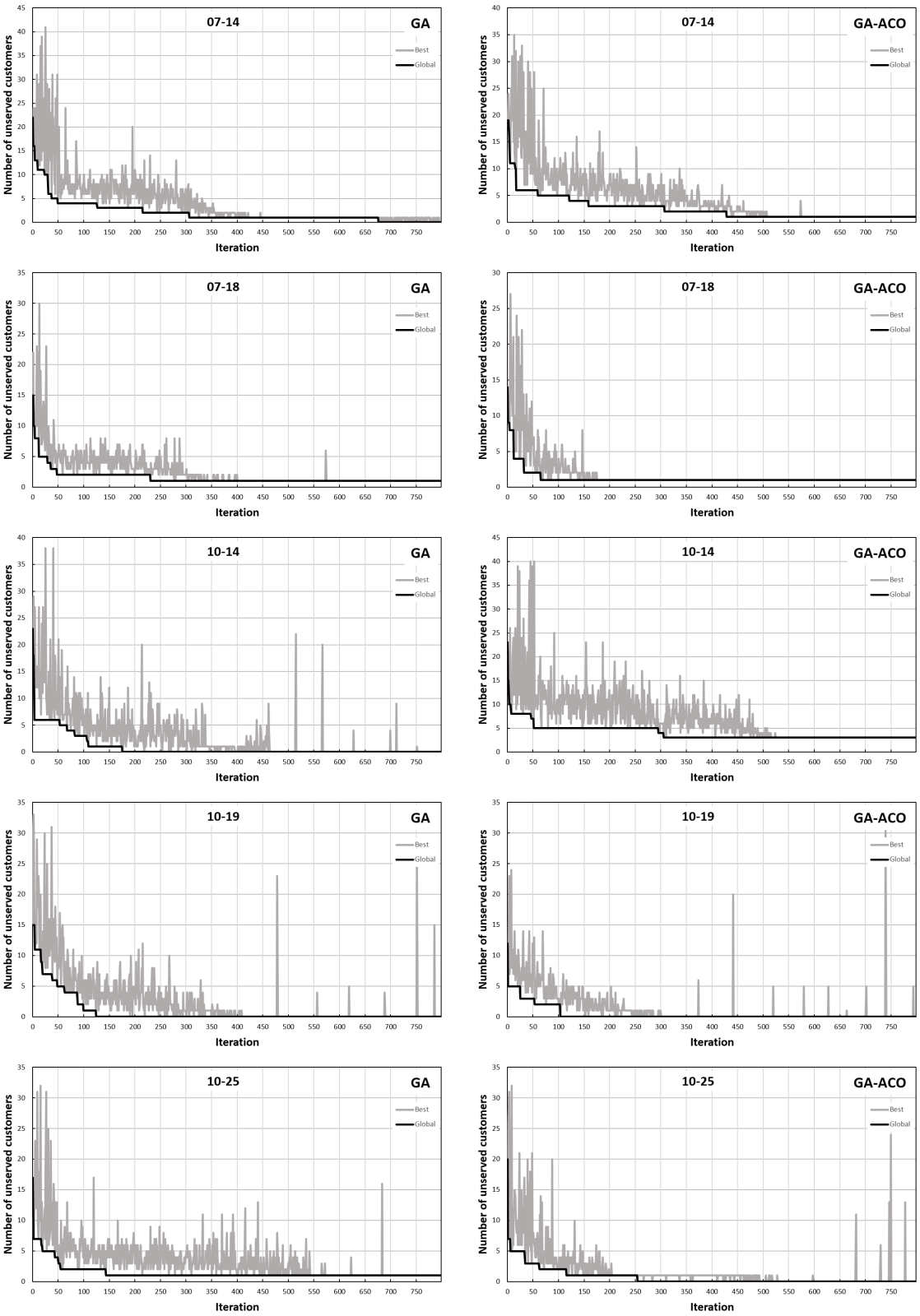

Figure 6.5: Number of customers not served in the best solution and in the global best solution for the **GA** approaches and 5 datasets.

Chapter 7

Conclusions and Future Work

This chapter summarizes the main conclusions of this dissertation and some future lines of work that will be part of the next research steps.

7.1 Conclusions

As explained previously, there is a wide range of real complex problems that cannot be solved with exact algorithms and therefore need heuristic approaches. This dissertation is centered on a swarm-based metaheuristic, Ant Colony Optimization, applied to different types of problems, mainly Resource Allocation Problems and Scheduling Problems. In order to be really efficient, ACO algorithms have to be combined with a local search that intensifies the search in a promising area of the search space.

The main goal of this thesis was the analysis and development of models and metaheuristics based on ACO algorithms and their application to real combinatorial optimization problems. This goal was pursued in all chapters comprising this dissertation. The main research questions presented in chapter 1 have been addressed in different chapters of this dissertation, the contribution of each of them for each question is presented here below:

G1 *To review the state of the art related to bio-inspired metaheuristics, mainly centered on Ant Colony Optimization, but including other approaches such as Genetic Algorithms and Harmony Search.*

Chapter 2. A wide review of bio-inspired metaheuristics has been included in this chapter. The metaheuristic based on ACO has been described with details together with two variants of it, the Elitist strategy and the Max-Min Ant System. The importance of a local search process for ACO has been explained, as has been the relationship between ACO and graphs. Several application fields of ACO have been presented to conclude this part. The second metaheuristic included in this chapter is GA. The different operators needed to implement GA have been described: selection, crossover, mutation and replacement. The importance of diversity and selection pressure in population-based metaheuristics has been explained. The last metaheuristic presented is HS, completed with a description of its parameters.

Chapter 5. The extension of GA to multi-objective problems has been included in this chapter, including the well-known NSGA-II, Non-dominated Sorting Genetic Algorithm II.

G2 *To review the state of the art related to Constraint Satisfaction Problems and how they can be solved with metaheuristics.*

Chapter 2. A detailed explanation of the main concepts of CSP has been presented in this chapter: variables, domains, constraints and assignments. Several classical examples have been given to illustrate those concepts: the n-Queens Problem, the Graph-Coloring Problem and the Sudoku Game. CSP has been extended to CSOP which includes an optimization objective. The Traveling Salesman Problem has been then depicted as an example of CSOP. Different resolution methods have been presented for both problem types, CSP and CSOP, including a solver that implements some of them. The ACO metaheuristic adapted to solve them has also been described in detail, including examples where ACO has been used to solve CSOP.

G3 *To review the state of the art in the different application fields analyzed in this dissertation: Resource Allocation Problems and Scheduling Problems, specially Course Allocation, Timetabling and Vehicle Routing Problems.*

Chapter 2. The RAP has been presented in this chapter with several examples of applications. The special case of the matching problem has been included and described with its main characteristics. Two examples of RAP have been given with details: the Course Allocation Problem that consists in allocating courses to students and the Course Timetabling Problem which consists in assigning timeslots and rooms to courses. The Multi-objective RAP has been explained with the Pareto front and the hypervolume metric that enables the comparison of different resolution methods for multi-objective problems. Several examples and resolution approaches have been presented for the MORAP. Finally the Vehicle Routing Problem has been described with some of its variants and different approaches used to solve it.

G4 *To develop different types of models for combinatorial optimization problems: Resource Allocation Problems and Scheduling Problems, specially Timetabling and Vehicle Routing Problems.*

Chapter 3. In this chapter a constraint-based model for the Course Timetabling Problem in Swiss secondary has been developed. This model includes the fact that students are allocated to classes at the same time as the timetable is built.

Chapter 4. In this chapter, two different types of problems existing at the Ecole Hôtelière de Lausanne have been analyzed and modeled. The first type is a simple Course Allocation Problem (CAP), and the second type combines a Course Allocation Problem with a Course Timetabling Problem (CAP&CTT). Both of them are modeled as CSOP.

Chapter 5. The model developed in this chapter is based on the models of chapter 3. This MORAP model is dedicated to the class allocation problem where students have to be split according to their profile while the resources needed have to be also considered.

Chapter 6. In this chapter, a model has been developed for a maVRP with many different constraints: time windows, heterogeneous fleet, multiple depots, multiple routes and incompatibilities.

Chapter 3.10. A model has been developed for the benchmark datasets of a Course Timetabling Problem (CTT).

G5 *To design new heuristic and metaheuristic approaches based on ACO algorithms.*

Chapter 3. We have presented two approaches based on ACO and on CSOP to solve the class allocation problem modeled as a resource allocation problem.

Chapter 4. For the simple CAP, we have compared two different approaches to solve the problem. The first approach, the Course Greedy Algorithm (CGA), allocates a course to the students sequentially. The second approach is based on CSOP and uses the solver Gecode. For the CAP&CTT, we have presented two approaches to solve it. The first approach uses a CSOP solver to find the solutions to both problems simultaneously and to each problem individually. The second approach uses the same CSOP solver for the CAP, but an ACO algorithm to solve the scheduling problem.

Chapter 5. In this chapter, we have presented three approaches to solve a many-objective optimization problem, based on ACO, HS and GA.

Chapter 6. In this chapter, five different approaches were presented to solve the maVRP, all based on classical metaheuristics, ACO and GA.

Chapter 3.10. An ACO approach has been designed to solve the CTT problem, it includes a local search based on a CSOP solver.

G6 *To hybridize metaheuristics and analyze their performance.*

Chapter 6. Two of the proposed approaches are hybrid metaheuristics that combine the two metaheuristics, ACO and GA. In one of them, ACO-GA, ACO is the main metaheuristic and GA is used for local search. In the other, GA-ACO, GA is the main metaheuristic and ACO is used to break ties instead of using randomness when a decision has to be made in order to complete the solution.

G7 *To apply those new approaches and metaheuristics to real industrial problems.*

Chapter 3. The heuristic and metaheuristic approaches proposed in this chapter have been applied to eight datasets coming from different Swiss schools.

Chapter 4.The proposed approaches are currently used at the Ecole Hôtelière de Lausanne to allocate the seats in the elective courses and to guarantee that this allocation is compatible with the timetable constraints.

Chapter 5. The three proposed approaches have been applied to the same eight datasets used in chapter 3.

Chapter 6. The proposed approaches have been applied to fifteen datasets coming from a logistics company.

G8 *To compare their performance in those problems.*

Chapter 3. Two approaches have been compared, one based on ACO and one based on CSOP. The ACO algorithm provides better solutions than the CSOP solver in a shorter time. As both of them use randomness for the allocation of resources, our results prove that the pheromones in the ACO approach help to find very good solutions in a much smaller amount of time. As the computation time is short for the ACO approach, the school's directors, who are in charge of the allocation of students and of the timetable, can simulate multiple scenarios of possible sets of profiles in their school.

Chapter 4. In the CAP, two metrics have been designed and used to quantify the quality of a solution: the Total Satisfaction Gap (TSG) analyzes the average level of satisfaction of the students and the Worst Satisfaction Gap (WSG) corresponds to the level of satisfaction of the worst off. For small instances, CSOP outperforms CGA for both metrics. For bigger instances, WSG is even better, but the computation time needed to improve TSG increases: TSG with CSOP is similar to TSG with CGA with 150 students. For the CAP&CTT, the two proposed approaches, CSOP and ACO, provide very good results in a reasonable computation time. The ACO approach is nevertheless much faster in finding good solutions.

Chapter 5. Two different methods have been used to compare the three approaches designed in this chapter, both of them based on the hypervolume indicator. The results show that NSGA-II outperforms the other approaches in four datasets out of eight. ACO outperforms the others in one dataset. For the other three datasets, the three approaches are competitive and provide good solutions.

Chapter 6. The results of the proposed approaches have been compared to two commercial tools looking at the quality of the solution and the computation time. The quality of the solution is measured by the number of served customers and the cost. Considering the number of served customers, one of the tools and the ACO-GA approach outperform the others. Considering the cost, GA and GA-ACO provide better results. The computation time is much better in GA and GA-ACO.

Chapter 3.10. The first results obtained with the ACO approach and a simple local search are encouraging, but far from optimized algorithms developed especially for the ITC-2007 datasets. Feasible solutions have always been found, that means with no hard constraint broken.

7.2 Future Work

In this section, a brief description is given about the future lines of work that could be done to continue and improve the current work and could be investigated in a new research work.

1. Bio-inspired Metaheuristics hybridizing
   A new line of work is related to the approaches proposed to solve the MORAP. Three metaheuristics have been used to solve the problem individually, ACO, HS and GA. Hybridizing those metaheuristics could improve their performance.

2. ACO tuning
   The ACO algorithm requires several parameters defined by the user to control the optimization (size of the colony, evaporation rate, balance between pheromones and visibility, variation of the pheromone bounds). Even if the tuning of those parameters is very often linked to the optimization problem itself, an analysis of the sensitivity of the approaches regarding the value of those parameters could lead to the definition of ranges in which the different problems would be solved efficiently.

3. Complete the MORAP with additional scheduling constraints
   In Chapter 5, the optimization problem of the educational system in Canton de Vaud has

been simplified by removing some constraints. To solve the complete problem, scheduling constraints have to be considered, such as teachers' and rooms' availability. This might lead to a much more complex problem and the computation time might then become an issue that would have to be tackled.

4. ACO approach time efficiency
   In most optimization problems and especially with large datasets, computation time becomes a key factor. For some applications, in particular industrial applications, the algorithm must be as fast as possible. In ACO, each ant builds its solution independently from the other ants. It would be interesting to parallelize the solution construction performed by the ants.

5. Complete the comparison with benchmark datasets for the CTT problem
   The ACO approach presented in chapter 3.10 to solve the CTT problem has to be further developed and improved in order to become competitive with other approaches presented in the literature.

Chapter 8

Conclusiones y Trabajo Futuro

Este capítulo resume las principales conclusiones de esta tesis y algunas líneas de trabajo futuras que serán parte de los próximos pasos de investigación.

8.1 Conclusions

Como se explicó anteriormente, existe una amplia gama de problemas complejos reales que no pueden ser resueltos con algoritmos exactos y por lo tanto necesitan enfoques heurísticos para ser resueltos. Esta tesis se centra en una metaheurística basada en enjambre, Ant Colony Optimization (ACO), aplicada a diferentes tipos de problemas, principalmente problemas de asignación de recursos (Resource Allocation) y problemas de planificación con recursos (Scheduling). Para ser realmente eficientes, los algoritmos ACO tienen que combinarse con una búsqueda local que intensifique la búsqueda en un área prometedora del espacio de soluciones.

El objetivo principal de esta tesis ha sido el análisis y desarrollo de modelos y de metaheurísticas basadas en algoritmos ACO y su aplicación a problemas reales de optimización combinatoria. Este objetivo se mantuvo a lo largo de los diferentes capítulos que componen esta tesis. Las principales preguntas de investigación presentadas en el capítulo 1 se han tratado en diferentes capítulos de esta disertación, la contribución de cada uno de estos capítulos a cada pregunta se presenta a continuación:

G1 *Revisar el estado del arte relacionado con metaheurísticas bio-inspiradas, centradas principalmente en Ant Colony Optimization, pero incluyendo otros enfoques como Genetic Algorithm (GA) y Harmony Search (HS).*

Capítulo 2. En este capítulo se ha incluido una amplia revisión de las metaheurísticas bio-inspiradas. La metaheurística basada en ACO ha sido descrita en detalle junto con dos variantes, Elitist y Max-Min Ant System. Se ha explicado la importancia de un proceso de búsqueda local para ACO, así como la relación entre ACO y grafos. Para concluir esta parte, se han presentado varios campos de aplicaciones de ACO. La segunda metaheurística incluida en este capítulo es GA. Se han descrito los diferentes operadores necesarios para implementar un GA: Selection, Crossover, Mutation y Replacement. Se ha explicado la

importancia de la diversidad y de la presión de selección en las metaheurísticas basadas en poblaciones. La última metaheurística presentada es HS descrita con sus parámetros.

Capítulo 5. La extensión de GA a problemas multi-objetivo ha sido descrita en este capítulo, incluyendo el bien conocido NSGA-II, Non-dominated Sorting Genetic Algorithm II.

G2 *Revisar el estado del arte relacionado con Constraint Satisfaction Problems (CSP) y cómo pueden ser resueltos con metaheurísticas.*

Capítulo 2. Una explicación detallada de los principales conceptos de los CSP se ha presentado en este capítulo: variables, dominios, restricciones y asignaciones. Se han dado varios ejemplos clásicos para ilustrar estos conceptos: n-Queens Problem, Graph-Coloring Problem y Sudoku Game. El CSP se ha ampliado al CSOP, que incluye un objetivo de optimización. El Traveling Salesman Problem se ha presentado como un ejemplo de CSOP. Se han incluido distintos métodos de resolución para ambos tipos de problemas, CSP y CSOP, así como un solver que implementa algunos de esos métodos. También se ha descrito en detalle la metaheurística ACO adaptada para ese tipo de modelos, incluyendo ejemplos en los que ACO se ha utilizado para resolver CSOP.

G3 *Revisar el estado del arte en los diferentes campos de aplicación analizados en esta disertación: Resource Allocation Problems y Scheduling Problems, especialmente asignación de cursos, planificación de horarios y problemas de enrutamiento de vehículos.*

Capítulo 2. El RAP se ha presentado en este capítulo con varios ejemplos de aplicación. El caso especial del problema de matching ha sido incluido y descrito con sus principales características. Dos ejemplos de RAP se han expuesto en detalle: el Course Allocation Problem que consiste en asignar cursos a estudiantes y el Course Timetabling Problem que consiste en asignar horarios y salas a cursos. También se ha descrito el RAP multi-objetivo junto con el frente de Pareto y el indicador de hipervolumen que permite la comparación de diferentes métodos de resolución de problemas multi-objetivo. Se han presentado varios ejemplos y enfoques de resolución para el MORAP. Para finalizar el capítulo, se ha descrito el Vehicle Routing Problem (VRP) con algunas de sus variantes y diferentes enfoques utilizados para resolverlo.

G4 *Desarrollar distintos modelos para problemas de optimización combinatoria: Resource Allocation Problems y Scheduling Problems, en particular Timetabling y Vehicle Routing Problems.*

Capítulo 3. En este capítulo se ha desarrollado un modelo basado en restricciones para el Course Timetabling Problem que existe en los colegios suizos. Este modelo incluye el hecho de que se asignan los estudiantes a las clases al mismo tiempo que se construye el horario.

Capítulo 4. En este capítulo, se han analizado y modelado dos tipos diferentes de problemas que existen en la Ecole Hôtelière de Lausanne. El primer tipo es un simple Course Allocation Problem (CAP), y el segundo tipo combina un Course Allocation Problem con un Course Timetabling Problem (CAP&CTT). Ambos se han modelado como CSOP.

Capítulo 5. El modelo utilizado en este capítulo se basa en los modelos desarrollados en el capítulo 3. Este modelo MORAP se centra en el problema de asignación de clases en el que los estudiantes han de repartirse según su perfil y considerando el número de recursos necesarios.

Capítulo 6. En este capítulo, se ha desarrollado un modelo para el maVRP con varias características diferentes: ventanas de tiempo, flota heterogénea, múltiples depósitos, múltiples rutas e incompatibilidades.

Capítulo 3.10. Se ha desarrollado un modelo para los conjuntos de datos benchmark de un Course Timetabling Problem.

G5 *Diseñar nuevos enfoques heurísticos y metaheurísticas basados en ACO.*

Capítulo 3. Se han presentado dos enfoques basados en ACO y en CSOP para resolver el problema de asignación de clases modelado como un problema de asignación de recursos.

Capítulo 4. Para el CAP simple, se han comparado dos enfoques de resolución diferentes. El primero, el Course Greedy Algorithm (CGA), asigna los cursos a estudiantes de forma secuencial. El segundo enfoque se basa en CSOP y utiliza el solver Gecode. Para el CAP&CTT, se han presentado dos enfoques para resolverlo. El primero utiliza un solver CSOP para encontrar las soluciones a ambos problemas simultáneamente y a cada problema individualmente. El segundo enfoque utiliza el mismo solver CSOP que el CAP, pero un algoritmo ACO para resolver el Scheduling Problem.

Capítulo 5.En este capítulo, hemos presentado tres enfoques para resolver un problema de optimización de multi-objetivo, basados en ACO, HS y GA.

Capítulo 6. En este capítulo, se han diseñado cuatro enfoques diferentes para resolver el maVRP, basados en las metaheurísticas clásicas, ACO y AG.

Capítulo 3.10. Se ha propuesto un enfoque ACO para resolver el problema CTT, que incluye una búsqueda local basada en un solver CSOP.

G6 *Hibridar metaheurísticas y analizar su rendimiento.*

Capítulo 6. Se han hibridado ACO y GA de dos formas distintas para resolver el VRP, en una de ellas GA es una búsqueda local de ACO, en la otra ACO influye en la toma de decisiones de GA y reduce la parte aleatoria.

G7 - *Aplicar estos métodos a problemas reales.*

Capítulo 3. Los enfoques heurísticos y metaheurísticos propuestos en este capítulo se han aplicado a ocho conjuntos de datos procedentes de varios colegios suizos.

Capítulo 4. Las estrategias propuestas se utilizan actualmente en la Ecole Hôtelière de Lausanne para asignar las plazas en las óptativas y garantizar que esta asignación sea compatible con una planificación horaria.

Capítulo 5. Los tres enfoques propuestos se han aplicado a los ocho conjuntos de datos utilizados en el capítulo 3.

Capítulo 6. Las estrategias propuestas se han aplicado a quince conjuntos de datos procedentes de una empresa de logística.

G8 *Comparar el rendimiento de estos enfoques en estos problemas.*

Capítulo 3. Se han comparado dos enfoques, uno basado en ACO y uno basado en CSOP. El algoritmo ACO proporciona mejores soluciones que el solver CSOP en un tiempo más corto. Como ambos asignan los recursos de forma aleatoria, nuestros resultados demuestran que las feromonas en ACO ayudan a encontrar muy buenas soluciones en un tiempo mucho menor. Como el tiempo de cómputo es corto para ACO, los directores de colegio, a cargo

de la asignación de estudiantes y del calendario, pueden simular múltiples escenarios de posibles conjuntos de perfiles en su escuela.

Capítulo 4. En el CAP se han diseñado y utilizado dos indicadores para cuantificar la calidad de una solución: el Total Satisfaction Gap (TSG) analiza el nivel medio de satisfacción de los estudiantes y el Worst Satisfaction Gap (WSG) corresponde al nivel de satisfacción del estudiante más perjudicado. Para pequeñas instancias, CSOP da mejores resultados para ambas métricas que CGA. Para instancias más grandes, WSG es todavía mejor, pero el tiempo de cálculo para mejorar TSG aumenta: TSG con CSOP es similar a TSG con CGA con instancias de 150 estudiantes. Para el CAP&CTT, los dos enfoques propuestos, CSOP y ACO, proporcionan muy buenos resultados en un tiempo de cómputo razonable. Sin embargo, ACO es mucho más rápido para encontrar buenas soluciones.

Capítulo 5. Se han utilizado ocho conjuntos de datos y dos métodos diferentes para comparar los tres enfoques diseñados en este capítulo, ambos basados en el indicador de hipervolumen. Los resultados muestran que NSGA-II supera a los otros enfoques en cuatro conjuntos de datos. ACO supera a los demás en un conjunto de datos. Para los otros tres, las tres estrategias son competitivas y proporcionan buenas soluciones.

Capítulo 6. Los resultados de las estrategias propuestas se han comparado con dos herramientas comerciales a nivel de la calidad de la solución y del tiempo de cálculo. La calidad de la solución se mide con el número de clientes que no pueden ser atendidos y el coste. Para maximizar el número de clientes atendidos, una de las herramientas y ACO-GA superan a los demás. Para la minimización del coste, GA y GA-ACO proporcionan mejores resultados. El tiempo de cálculo es mucho mejor en GA y GA-ACO.

Capítulo 3.10. Los primeros resultados obtenidos con el enfoque ACO y una simple búsqueda local son prometedores, pero lejos de algoritmos específicos y optimizados desarrollados especialmente para los conjuntos de datos ITC-2007. Siempre se han encontrado soluciones viables, lo que significa que no se violan restricciones duras.

8.2 Trabajo futuro

En esta sección se proporciona una breve descripción de las futuras líneas de trabajo previstas para continuar y mejorar el trabajo actual y que podrían desarollarse en un nuevo trabajo de investigación.

1. Hibridar metaheurísticas bio-inspiradas
   Una nueva línea de trabajo está relacionada con las estrategias propuestas para resolver el MORAP. Se han utilizado tres metaheurísticas independientes para resolver el problema, ACO, HS y GA. Hibridar esas metaheurísticas podría mejorar su rendimiento.

2. Ajustar ACO
   El algoritmo ACO requiere varios parámetros definidos por el usuario para controlar la optimización (tamaño de la colonia, evaporación, equilibrio entre feromonas y visibilidad, variación de los límites de feromonas). El ajuste de estos parámetros está muy relacionado con el propio problema de optimización, sin embargo un análisis de sensibilidad de las estrategias ACO al valor de esos parámetros podría conducir a la definición de rangos que permitirían una resolución eficiente de los problemas analizados en esta tesis.

3. Completar el MORAP con restricciones de programación de recursos adicionales
   En el Capítulo 5, el problema de optimización del sistema educativo del Cantón de Vaud ha sido simplificado eliminando algunas restricciones. Para resolver el problema completo, hay que considerar las restricciones de programación, como la disponibilidad de profesores y salas. Esto podría conducir a un problema mucho más complejo y el tiempo de cálculo podría convertirse en un problema que habría que abordar.

4. Tiempo de cálculo en ACO
   En la mayoría de los problemas de optimización y especialmente con conjuntos de datos grandes, el tiempo de cálculo se convierte en un factor clave. Para algunas aplicaciones, en particular las industriales, el algoritmo debe ser lo más rápido posible. En ACO, cada hormiga construye su solución independientemente de las otras hormigas. Sería interesante paralelizar la construcción de la solución realizada por las hormigas.

5. Completar el desarrollo y la comparación con los conjuntos de datos benchmark del problema CTT
   El enfoque ACO presentado en el capítulo 3.10 para resolver el problema de CTT tiene que desarrollarse y mejorarse para ser competitivo con otras estrategias presentadas en la literatura.

Appendix A

Notation used in Chapter 3

Tables A.1 and A.2 contains the list of the notations used for the data of the models presented in Chapter 3 and table A.3 is used for the output results. The results are either the results of the allocations or sets that are deduced from the allocations.

Notation	Description
P_{max}	Max. number of lessons per day per student
W_{min}	Objective: Min. weight per day per student
W_{max}	Objective: Max. weight per day per student
$d \in D$	Day
$p \in P$	Period
$PD(p)$	Day of p
$PHF(p) \in \{0,1\}$	=1 if p is the first period of a half-day
$PHL(p) \in \{0,1\}$	=1 if p is the last period of a half-day
$PQF(p) \in \{0,1\}$	=1 if p is the first period of a quarter-day
$PQL(p) \in \{0,1\}$	=1 if p is the last period of a quarter-day
$PN(p)$	Period(s) before or after p in the quarter-day
$k \in K$	Categories
$KG(k)$	Grade of k
$KT(k)$	Type of k $\in \{section, OS, OCOM, level\}$
$KS_{max}(k)$	Max. number of students in a class of k
$q \in Q$	Classes
$QK(q)$	Category of class q
$r \in R$	Rooms
$RT \in \{0,1\}$	= 1 if r requires travel time

Table A.1: Notation: Input data - Part I.

Notation	Description
$c \in C$	Courses
$CQ(c)$	Class of c
$CR(c)$	List of adapted rooms for c
$CT(c)$	List of skilled teachers for c
$CD_{max}(c)$	Max. number of lessons per day for c
$CW(c)$	Weight of c
$CF(c) \subset C$	List of forbidden neighbors of c
$CN(c) \subset C$	List of desirable neighbors of c
$l \in L$	Lessons
$LC(l) \in C$	Course of l
$t \in T$	Teachers
$TW_{max}(t)$	Max. number of working periods
$TW_{min}(t)$	Min. number of working periods
$TP(t) \subseteq P$	Periods when t is available
$s \in S$	Students
$SK(s) \subset K$	List of categories to which s belongs

Table A.2: Notation: Input data - Part II.

Notation	Description
Assignment	
$\mu_k(s) \in Q$	Class assigned to s for $k \in SK(s)$
$\mu_R(l) \in R$	Room assigned to l
$\mu_T(l) \in T$	Teacher assigned to l
$\mu_P(l) \in P$	Period scheduled for l
Deduced sets	
$\nu_L(s)$	List of lessons assigned to s $= \{l \in \nu_L(s) : PD(\mu_P(l)) = d\}$
$\nu_{LD}(s,d)$	List of lessons assigned to s on day d $= \{l \in \nu_L(s) : PD(\mu_P(l)) = d\}$
$\nu_W(s,d)$	Weight of lessons assigned to s on day d $= \sum_{l \in \nu_{LD}(s,d)} CW(LC(l))$
$\nu_S(q,k)$	Number of students assigned to q that belong to k $= Card(\{s \in S : \mu_k(s) = q\})$

Table A.3: Notation: Output results.

Appendix B

Complete Results for the maVRP

This appendix contains all the charts for the maVRP presented in Chapter 6 and complete the last part of Section 6.7.3.3.

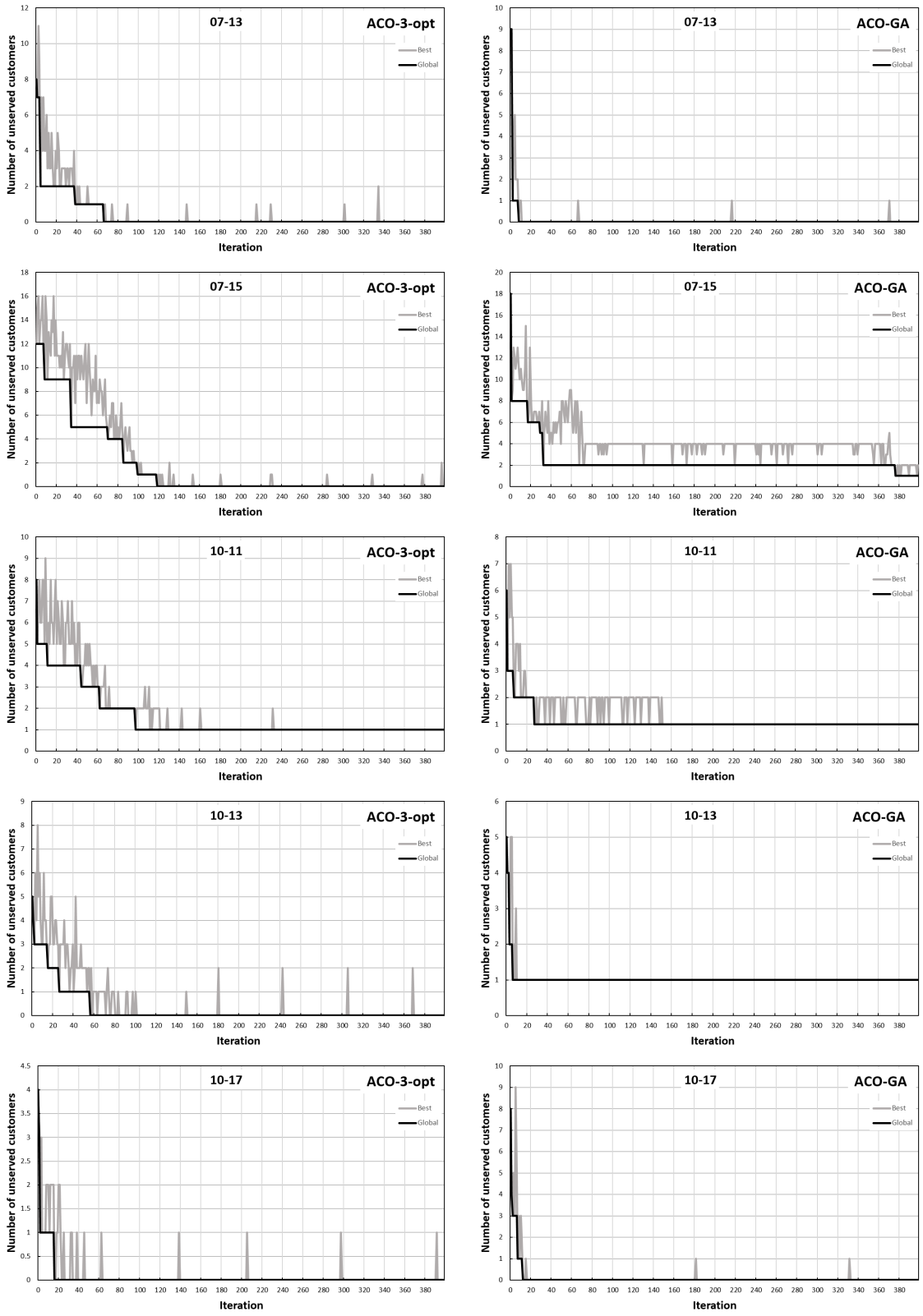

Figure B.1: Number of customers not served in the best solution and in the global best solution for the **ACO** approaches and 5 datasets.

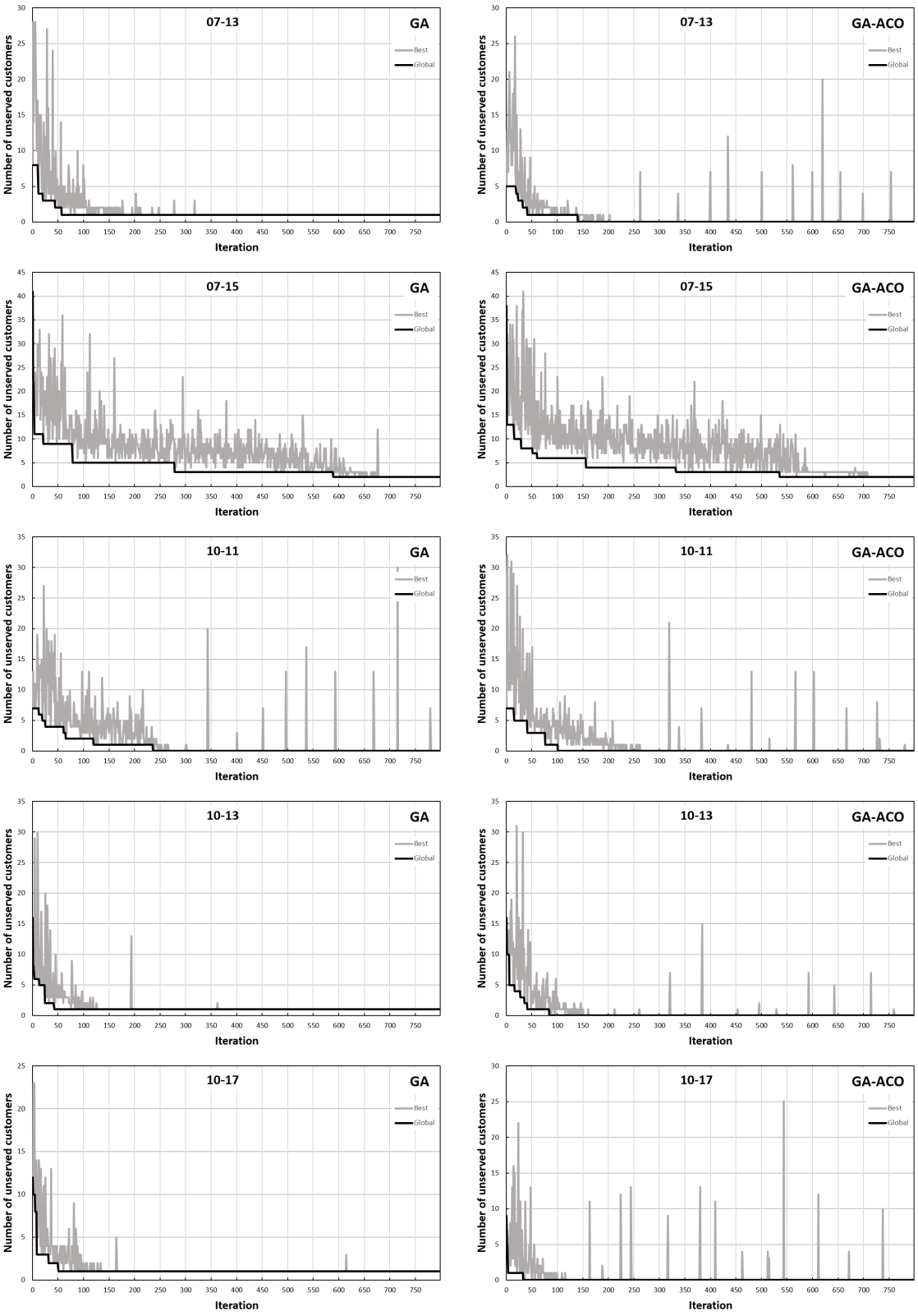

Figure B.2: Number of customers not served in the best solution and in the global best solution for the **GA** approaches and 5 datasets.

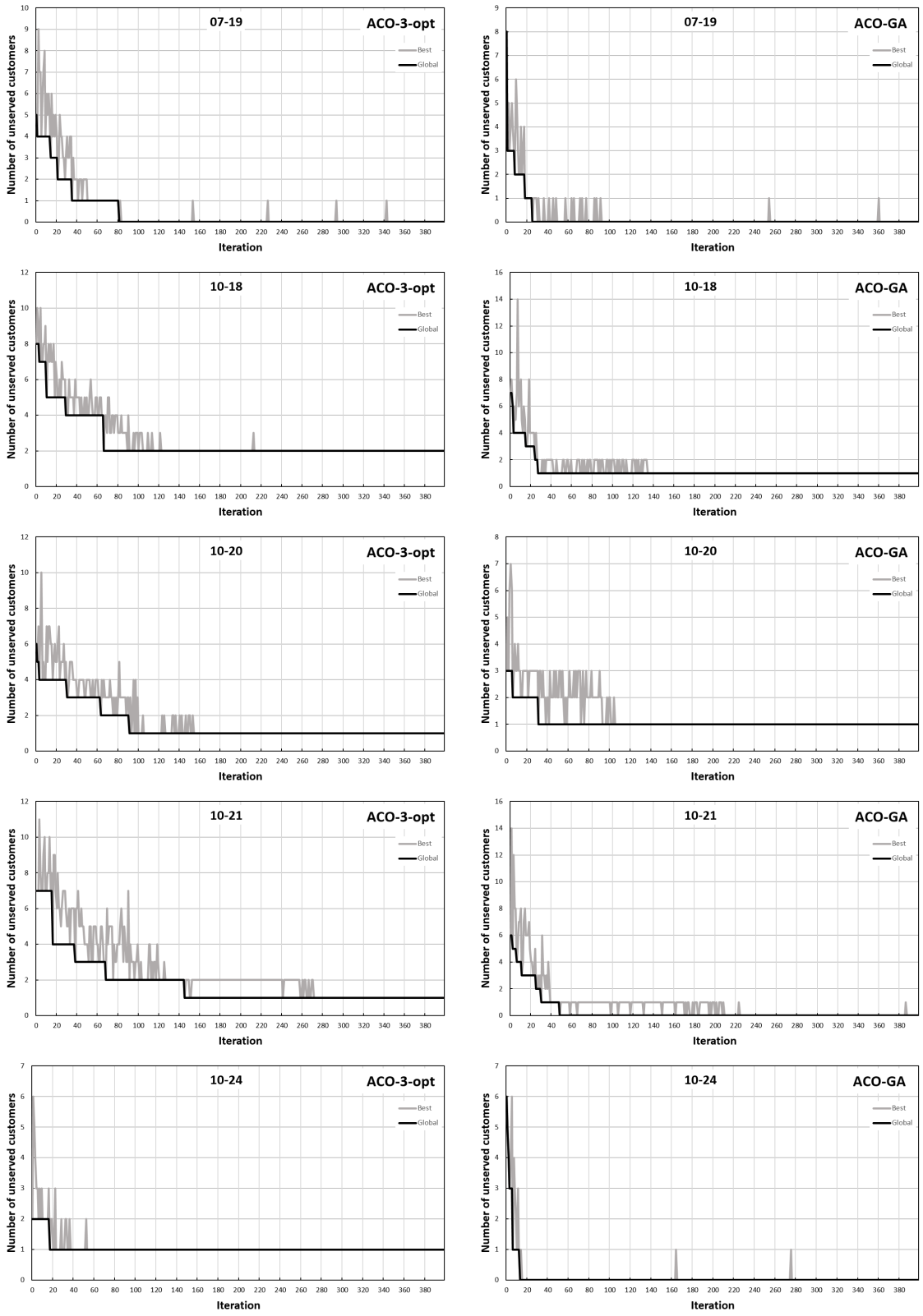

Figure B.3: Number of customers not served in the best solution and in the global best solution for the **ACO** approaches and 5 datasets.

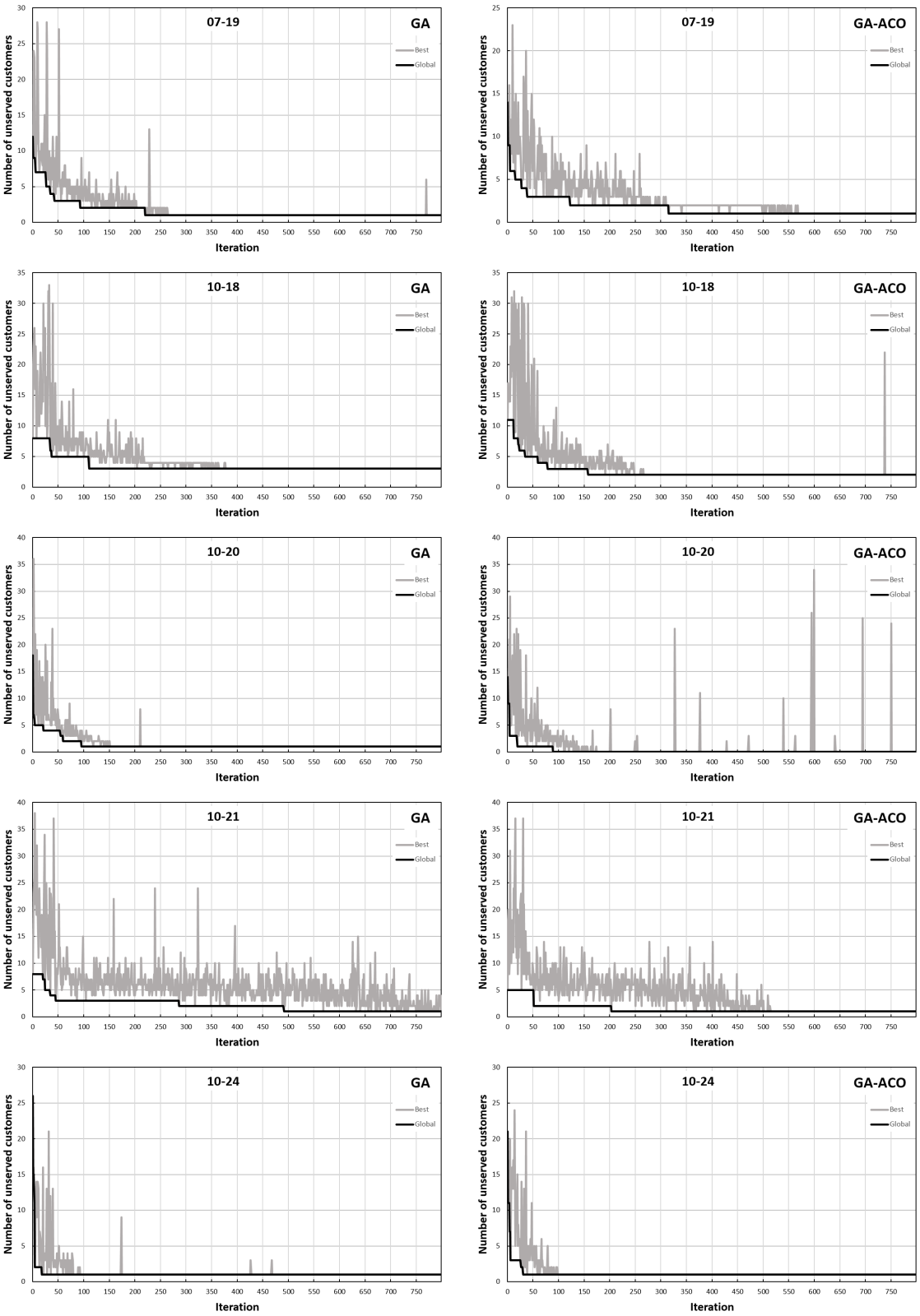

Figure B.4: Number of customers not served in the best solution and in the global best solution for the **GA** approaches and 5 datasets.

Bibliography

[1] L. E. Agustín-Blas, S. Salcedo-Sanz, E. G. Ortíz-García, A. Portilla-Figueras, and Á. M. Pérez-Bellido. A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups. *Expert Syst. Appl.*, 36(3):7234–7241, 2009.

[2] I. Alaya, C. Solnon, and K. Ghedira. Ant colony optimization for multi-objective optimization problems. In *ICTAI (1)*, pages 450–457. Citeseer, 2007.

[3] C. H. Antunes, P. Lima, E. Oliveira, and D. F. Pires. A multi-objective simulated annealing approach to reactive power compensation. *Engineering Optimization*, 43(10):1063–1077, 2011.

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[5] M. Avci and S. Topaloglu. A hybrid metaheuristic algorithm for heterogeneous vehicle routing problem with simultaneous pickup and delivery. *Expert Systems with Applications*, 53:160–171, 2016.

[6] H. Babaei, J. Karimpour, and A. Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86:43–59, 2015.

[7] B. M. Baker and M. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[8] A. Basso and L. A. Peccati. Optimal resource allocation with minimum activation levels and fixed costs. *European Journal of Operational Research*, 131(3):536–549, 2001.

[9] G. N. Beligiannis, C. N. Moschopoulos, G. P. Kaperonis, and S. D. Likothanassis. Applying evolutionary computation to the school timetabling problem: The greek case. *Computers & Operations Research*, 35(4):1265–1280, 2008.

[10] J. E. Bell and P. R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced engineering informatics*, 18(1):41–48, 2004.

[11] F. Belmecheri, C. Prins, F. Yalaoui, and L. Amodeo. Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *Journal of intelligent manufacturing*, 24(4):775–789, 2013.

[12] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[13] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[14] M. R. Bonyadi and Z. Michalewicz. Particle swarm optimization for single objective continuous space problems: a review, 2017.

[15] I. BoussaïD, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[16] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[17] E. Budish and E. Cantillon. The multi-unit assignment problem: Theory and evidence from course allocation at harvard. *The American economic review*, 102(5):2237–2271, 2012.

[18] E. B. Budish and E. Cantillon. The multi-unit assignment problem: Theory and evidence from course allocation at harvard. 2010.

[19] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In *Meta-heuristics*, pages 285–296. Springer, 1999.

[20] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):32, 2015.

[21] J. I. Cano, L. Sánchez, D. Camacho, E. Pulido, and E. Anguiano. Allocation of educational resources through happiness maximization. In *Proceedings of the 4th International Conference on Software and Data Technologies*, 2009.

[22] J. I. Cano, L. Sánchez, D. Camacho, E. Pulido, and E. Anguiano. Using preferences to solve student–class allocation problem. In *Intelligent Data Engineering and Automated Learning-IDEAL 2009*, pages 626–632. Springer, 2009.

[23] S. K. Chaharsooghi and A. H. M. Kermani. An effective ant colony optimization algorithm (aco) for multi-objective resource allocation problem (morap). *Applied Mathematics and Computation*, 200(1):167–177, 2008.

[24] S. Chand and M. Wagner. Evolutionary many-objective optimization: A quick-start guide. *Surveys in Operations Research and Management Science*, 20(2):35–42, 2015.

[25] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on evolutionary computation*, 8(3):256–279, 2004.

[26] T. Cooper and J. Kingston. The complexity of timetable construction problems. *Practice and Theory of Automated Timetabling*, pages 281–295, 1996.

[27] J.-F. Cordeau and M. Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, 2012.

[28] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the operational research society*, 48(3):295–305, 1997.

[29] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[30] C. de Vaud. Contenus d'enseignement plan d'études romand (per), 2016.

[31] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[32] K. Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011.

[33] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International Conference on Parallel Problem Solving From Nature*, pages 849–858. Springer, 2000.

[34] D. Dechampai, L. Tanwanichkul, K. Sethanan, and R. Pitakaso. A differential evolution algorithm for the capacitated vrp with flexibility of mixing pickup and delivery services and the maximum duration of a route in poultry industry. *Journal of Intelligent Manufacturing*, pages 1–20, 2015.

[35] J. Del Ser, M. N. Bilbao, C. Perfecto, and S. Salcedo-Sanz. A harmony search approach for the selective pick-up and delivery problem with delayed drop-off. In *Harmony Search Algorithm*, pages 121–131. Springer, 2016.

[36] U. Derigs and M. Pullmann. Solving multitrip vehicle routing under order incompatibilities: A vrp arising in supply chain management. *Networks*, 64(1):29–39, 2014.

[37] G. Desaulniers and Q. Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with pickup and delivery*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[38] L. Di Gaspero. Integration of metaheuristics and constraint programming. In *Springer Handbook of Computational Intelligence*, pages 1225–1237. Springer, 2015.

[39] L. Di Gaspero, B. McCollum, and A. Schaerf. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical report, Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen's University, Belfast, United Kingdom, 2007.

[40] L. Di Gaspero, A. Rendl, and T. Urli. A hybrid aco+ cp for balancing bicycle sharing systems. In *International Workshop on Hybrid Metaheuristics*, pages 198–212. Springer, 2013.

[41] F. Diebold, H. Aziz, M. Bichler, F. Matthes, and A. Schneider. Course allocation via stable matching. *Business & Information Systems Engineering*, 6(2):97–110, 2014.

[42] R. Dondo and J. Cerdá. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(3):1478–1507, 2007.

[43] M. Dorigo and M. Birattari. Ant colony optimization. In *Encyclopedia of machine learning*, pages 36–39. Springer, 2010.

[44] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[45] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[46] K. A. Dowsland and J. M. Thompson. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156(3):313–324, 2008.

[47] M. Ehrgott, K. Klamroth, and C. Schwehm. An mcdm approach to portfolio optimization. *European Journal of Operational Research*, 155(3):752–770, 2004.

[48] E. Fallah-Mehdipour, O. B. Haddad, M. M. R. Tabari, and M. A. Mariño. Extraction of decision alternatives in construction management projects: Application and adaptation of nsga-ii and mopso. *Expert Systems with Applications*, 39(3):2794–2803, 2012.

[49] K. Fan, W. You, and Y. Li. An effective modified binary particle swarm optimization (mbpso) algorithm for multi-objective resource allocation problem (morap). *Applied Mathematics and Computation*, 221:257–267, 2013.

[50] F.-A. Fortin and M. Parizeau. Revisiting the nsga-ii crowding-distance computation. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 623–630. ACM, 2013.

[51] R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[52] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[53] L. M. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the operational research society*, pages 167–176, 1999.

[54] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[55] Z. W. Geem. State-of-the-art in the structure of harmony search algorithm. In *Recent Advances In Harmony Search Algorithm*, pages 1–10. Springer, 2010.

[56] Z. W. Geem, J. H. Kim, and G. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.

[57] S. Ghorbani and M. Rabbani. A new multi-objective algorithm for a project selection problem. *Advances in Engineering Software*, 40(1):9–14, 2009.

[58] F. Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[59] F. P. Goksal, I. Karaoglan, and F. Altiparmak. A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers & Industrial Engineering*, 65(1):39–53, 2013.

[60] B. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

[61] A. González-Pardo, D. F. Barrero, D. Camacho, and M. D. R-Moreno. A case study on grammatical-based representation for regular expression evolution. In *Trends in Practical Applications of Agents and Multiagent Systems*, pages 379–386. Springer, 2010.

[62] A. Gonzalez-Pardo and D. Camacho. A new csp graph-based representation for ant colony optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 689–696. IEEE, 2013.

[63] A. Gonzalez-Pardo and D. Camacho. Environmental influence in bio-inspired game level solver algorithms. In *Intelligent Distributed Computing VII*, pages 157–162. Springer, 2014.

[64] A. Gonzalez-Pardo, F. Palero, and D. Camacho. An empirical study on collective intelligence algorithms for video games problem-solving. *Computing and Informatics*, 2015.

[65] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[66] K. Harwood, C. Mumford, and R. Eglese. Investigating the use of metaheuristics for solving single vehicle routing problems with time-varying traversal costs. *Journal of the Operational Research Society*, 64(1):34–47, 2013.

[67] L. He and S. He. Solving water resource scheduling problem through an improved artificial fish swarm algorithm. *International Journal of Simulation Modelling (IJSIMM)*, 14(1), 2015.

[68] C. Heimerl and R. Kolisch. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR spectrum*, 32(2):343–368, 2010.

[69] H. Hellkvist and W. Sjöstedt. Toward automated timetabling at teknat. 2012.

[70] E. Hoffman, J. Loessi, and R. Moore. Constructions for the solution of the m queens problem. *Mathematics Magazine*, 42(2):66–72, 1969.

[71] J. H. Holland. Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence. *Ann Arbor: University of Michigan Press, 1975*, 1, 1975.

[72] W. Hu, H. Liang, C. Peng, B. Du, and Q. Hu. A hybrid chaos-particle swarm optimization algorithm for the vehicle routing problem with time window. *Entropy*, 15(4):1247–1270, 2013.

[73] E. J. Hughes. Evolutionary many-objective optimisation: many once or one many? In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 222–227. IEEE, 2005.

[74] F. Jacobsen, A. Bortfeldt, and H. Gehring. Timetabling at german secondary schools: tabu search versus constraint programming. In *Proceedings 6th international conference on the practice and theory of automated timetabling, PATAT2006*, pages 439–442. Citeseer, 2006.

[75] H. Jain and K. Deb. An improved adaptive approach for elitist nondominated sorting genetic algorithm for many-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 307–321. Springer, 2013.

[76] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[77] S. Karakatič and V. Podgorelec. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing*, 27:519–532, 2015.

[78] M. Khichane, P. Albert, and C. Solnon. Integration of aco in a constraint programming language. In *International Conference on Ant Colony Optimization and Swarm Intelligence*, pages 84–95. Springer, 2008.

[79] B.-I. Kim and S.-J. Son. A probability matrix based particle swarm optimization for the capacitated vehicle routing problem. *Journal of Intelligent Manufacturing*, 23(4):1119–1126, 2012.

[80] J. H. Kingston. A tiling algorithm for high school timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 208–225. Springer, 2004.

[81] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[82] F. Kojima. Efficient resource allocation under multi-unit demand. *Games and Economic Behavior*, 82:1–14, 2013.

[83] A. L. Kok, C. M. Meyer, H. Kopfer, and J. M. J. Schutten. A dynamic programming heuristic for the vehicle routing problem with time windows and european community social legislation. *Transportation Science*, 44(4):442–454, 2010.

[84] I. Landa-Torres, D. Manjarres, S. Salcedo-Sanz, J. D. Ser, and S. Gil-Lopez. A multi-objective grouping harmony search algorithm for the optimal distribution of 24-hour medical emergency units. *Expert Syst. Appl.*, 40(6):2343–2349, 2013.

[85] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[86] G. Laporte, Y. Nobert, and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation science*, 22(3):161–172, 1988.

[87] J. K. Lenstra and A. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[88] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, 2004.

[89] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, 30(1):167–190, 2008.

[90] R. Lewis, B. Paechter, B. McCollum, et al. *Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition*. Cardiff Business School, 2007.

[91] Y.-C. Liang and C.-Y. Chuang. Variable neighborhood search for multi-objective resource allocation problems. *Robotics and Computer-Integrated Manufacturing*, 29(3):73–78, 2013.

[92] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[93] M. López-Ibánez, L. Paquete, and T. Stützle. Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms*, 5(1):111–137, 2006.

[94] M. López-Ibánez, T. Stützle, and M. Dorigo. Ant colony optimization: A component-wise overview. *Techreport, IRIDIA, Universite Libre de Bruxelles*, 2015.

[95] B. Maenhout and M. Vanhoucke. An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. *Omega*, 41(2):485–499, 2013.

[96] D. Manjarres, I. Landa-Torres, S. Gil-Lopez, J. Del Ser, M. N. Bilbao, S. Salcedo-Sanz, and Z. W. Geem. A survey on applications of the harmony search algorithm. *Engineering Applications of Artificial Intelligence*, 26(8):1818–1831, 2013.

[97] Y. Marinakis and M. Marinaki. A hybrid genetic–particle swarm optimization algorithm for the vehicle routing problem. *Expert Systems with Applications*, 37(2):1446–1455, 2010.

[98] I. Markov, S. Varone, and M. Bierlaire. Integrating a heterogeneous fixed fleet and a flexible assignment of destination depots in the waste collection vrp with intermediate facilities. *Transportation Research Part B: Methodological*, 84:256–273, 2016.

[99] M. Mavrovouniotis, F. M. Müller, and S. Yang. Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE Transactions on Cybernetics*, 2016.

[100] B. McCollum and N. Ireland. University timetabling: Bridging the gap between research and practice. *E Burke, HR, ed.: PATAT*, pages 15–35, 2006.

[101] K. Nekooei, M. M. Farsangi, H. Nezamabadi-Pour, and K. Y. Lee. An improved multi-objective harmony search for optimal placement of dgs in distribution systems. *IEEE Transactions on smart grid*, 4(1):557–567, 2013.

[102] D. W. K. Ng, E. S. Lo, and R. Schober. Multiobjective resource allocation for secure communication in cognitive radio networks with wireless information and power transfer. *IEEE Transactions on Vehicular Technology*, 65(5):3166–3184, 2016.

[103] A. M. Nogareda and D. Camacho. Integration of ant colony optimization algorithms with gecode. In *Principles and Practice of Constraint Programming (Doctoral Program CP 2014), International Conference on*, pages 59–64, 2014.

[104] A. M. Nogareda and D. Camacho. Constraint-based model design for timetabling problems in secondary schools. In *Innovations in Intelligent SysTems and Applications (INISTA), 2015 International Symposium on*, pages 1–6. IEEE, 2015.

[105] A.-M. Nogareda and D. Camacho. Optimizing satisfaction in a multi-courses allocation problem. In *Intelligent Distributed Computing IX*, pages 247–256. Springer, 2016.

[106] A.-M. Nogareda and D. Camacho. Optimizing satisfaction in a multi-courses allocation problem combined with a timetabling problem. *Soft Computing*, pages 1–10, 2016.

[107] A.-M. Nogareda and D. Camacho. A constraint-based approach for classes setting-up problems in secondary schools. *International Journal of Simulation Modelling (IJSIMM)*, 16(2), 2017.

[108] A.-M. Nogareda, D. Camacho, and J. Del Ser. A comparison of bio-inspired heuristics applied to many-objective resource allocation problems. *Soft Computing*, Submitted 2017.

[109] A.-M. Nogareda, J. Del Ser, and D. Camacho. Hybrid metaheuristics to manage complex vehicle routing problems. *Engineering Applications of Artificial Intelligence*, Submitted 2017.

[110] C. Nothegger, A. Mayer, A. Chwatal, and G. R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 194(1):325–339, 2012.

[111] U. of Twente. International Timetabling Competition 2011, 2014.

[112] B. Ombuki, B. J. Ross, and F. Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.

[113] M. Osman, M. A. Abo-Sinna, and A. Mousa. An effective genetic algorithm approach to multiobjective resource allocation problems (moraps). *Applied Mathematics and Computation*, 163(2):755–768, 2005.

[114] S. H. R. Pasandideh, S. T. A. Niaki, and S. Sharafzadeh. Optimizing a bi-objective multi-product epq model with defective items, rework and limited orders: Nsga-ii and mopso algorithms. *Journal of Manufacturing Systems*, 32(4):764–770, 2013.

[115] N. Pillay. A survey of school timetabling research. *Annals of Operations Research*, 218(1):261–293, 2014.

[116] S. Poles, Y. Fu, and E. Rigoni. The effect of initial population sampling on the convergence of multi-objective genetic algorithms. In *Multiobjective Programming and Goal Programming*, pages 123–133. Springer, 2009.

[117] G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngas, C. Nurmi, and D. Ranson. An xml format for benchmarks in high school timetabling. *Annals of Operations Research*, 194(1):385–397, 2012.

[118] C. Prins. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22(6):916–928, 2009.

[119] L. Rachmawati and D. Srinivasan. A hybrid fuzzy evolutionary algorithm for a multi-objective resource allocation problem. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 6–pp. IEEE, 2005.

[120] J. Rada-Vilela, M. Chica, Ó. Cordón, and S. Damas. A comparative study of multi-objective ant colony optimization algorithms for the time and space assembly line balancing problem. *Applied Soft Computing*, 13(11):4370–4382, 2013.

[121] M. Rahoual and R. Saad. Solving timetabling problems by hybridizing genetic algorithms and taboo search. In *6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 467–472, 2006.

[122] M. Randall. Solution approaches for the capacitated single allocation hub location problem using ant colony optimisation. *Computational Optimization and Applications*, 39(2):239–261, 2008.

[123] M. Reed, A. Yiannakou, and R. Evering. An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing*, 15:169–176, 2014.

[124] F. Rossi, K. B. Venable, and T. Walsh. A short introduction to preferences: between artificial intelligence and social choice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(4):1–102, 2011.

[125] M. Ryan. Constraint-based multi-robot path planning. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 922–928. IEEE, 2010.

[126] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and A. Portilla-Figueras. The coral reefs optimization algorithm: an efficient meta-heuristic for solving hard optimization problems. In *Proceedings of the 15th International Conference on Applied Stochastic Models and Data Analysis (ASMDA2013), Mataró*, pages 751–758, 2013.

[127] S. Salcedo-Sanz, D. Manjarres, Á. Pastor-Sánchez, J. Del Ser, J. A. Portilla-Figueras, and S. Gil-Lopez. One-way urban traffic reconfiguration using a multi-objective harmony search approach. *Expert Systems with Applications*, 40(9):3341–3350, 2013.

[128] S. Salcedo-Sanz, A. Pastor-Sánchez, J. Portilla-Figueras, and L. Prieto. Effective multi-objective optimization with the coral reefs optimization algorithm. *Engineering Optimization*, 48(6):966–984, 2016.

[129] R. Santiago-Mozos, S. Salcedo-Sanz, M. DePrado-Cumplido, and C. Bousoño-Calzón. A two-phase heuristic evolutionary algorithm for personalizing course timetables: a case study in a spanish university. *Computers & operations research*, 32(7):1761–1776, 2005.

[130] K. Sastry, D. E. Goldberg, and G. Kendall. Genetic algorithms. In *Search methodologies*, pages 93–117. Springer, 2014.

[131] C. Schulte, G. Tack, and M. Z. Lagerkvist. Modeling and programming with gecode, 2010.

[132] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In *International Workshop on Ant Algorithms*, pages 1–13. Springer, 2002.

[133] K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Applications of evolutionary computing*, pages 334–345. Springer, 2003.

[134] C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In *ECAI*, pages 118–122. Citeseer, 2000.

[135] C. Solnon. Ants can solve constraint satisfaction problems. *Evolutionary Computation, IEEE Transactions on*, 6(4):347–357, 2002.

[136] C. Solnon. *Ant colony optimization and constraint programming*. Wiley Online Library, 2010.

[137] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[138] T. Sönmez and M. U. Ünver. Course bidding at business schools*. *International Economic Review*, 51(1):99–123, 2010.

[139] T. Sönmez, M. U. Ünver, et al. Matching, allocation, and exchange of discrete resources. *Handbook of social Economics*, 1:781–852, 2011.

[140] T. Stutzle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314. IEEE, 1997.

[141] T. Stützle and H. H. Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.

[142] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 17(4):441–449, 2001.

[143] I. X. Tassopoulos and G. N. Beligiannis. Solving effectively the school timetabling problem using particle swarm optimization. *Expert Systems with Applications*, 39(5):6029–6040, 2012.

[144] I. X. Tassopoulos and G. N. Beligiannis. Using particle swarm optimization to solve effectively the school timetabling problem. *Soft Computing*, 16(7):1229–1252, 2012.

[145] N. E. Toklu, L. M. Gambardella, and R. Montemanni. A multiple ant colony system for a vehicle routing problem with time windows and uncertain travel times. *Journal of Traffic and Logistics Engineering*, 2(1), 2014.

[146] P. Toth and D. Vigo. Branch-and-bound algorithms for the capacitated vrp. In *The vehicle routing problem*, pages 29–51. Society for Industrial and Applied Mathematics, 2001.

[147] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[148] E. Tsang. *Foundations of Constraint Satisfaction: The Classic Text*. BoD–Books on Demand, 2014.

[149] F. Vavak and T. C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 192–195. IEEE, 1996.

[150] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

[151] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[152] G. J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

[153] B. Yu, Z.-Z. Yang, and B. Yao. An improved ant colony optimization for vehicle routing problem. *European journal of operational research*, 196(1):171–176, 2009.

[154] H. Zhang, C. Jiang, N. C. Beaulieu, X. Chu, X. Wang, and T. Q. Quek. Resource allocation for cognitive small cell networks: A cooperative bargaining game theoretic approach. *IEEE Transactions on Wireless Communications*, 14(6):3481–3493, 2015.

[155] T. Zhang, W. A. Chaovalitwongse, and Y. Zhang. Integrated ant colony and tabu search approach for time dependent vehicle routing problems with simultaneous pickup and delivery. *Journal of Combinatorial Optimization*, 28(1):288–309, 2014.

[156] E. Zitzler. Evolutionary algorithms for multiobjective optimization: Methods and applications. 1999.