$N°$ d'ordre : 05/2020_D/INF

République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique
Départment d'Informatique

Thèse De Doctorat en Sciences

Présentée pour l'obtention du grade de DOCTEUR

En INFORMATIQUE

Spécialité : Systèmes Intélligents et Ingénierie du Logiciel

Par Mr. Hakim Mitiche

Approches Métaheuristiques pour l'Allocation de Tâches dans les Systèmes Multi-Agents

Soutenue publiquement, le 23/01/2020, devant le jury composé de :

Mr. AZZOUNE Hamid	Professeur à l'USTHB	Président
Mme. BOUGHACI Dalila	Professeur à l'USTHB	Directrice de thèse
Mme. BENATCHBA Karima	Professeur à l'E.S.I	Examinatrice
Mr. BOULIF Menaouar	Professeur à l'UMBB (Boumerdes)	Examinateur
Mr. AIT-ZAI Abdelhakim	Professeur à l'USTHB	Examinateur
Mr. KAZAR Okba	Professeur à l'UMKB.(Biskra)	Examinateur
Mme. GINI Maria	Professeur à U. Minnesota (USA)	Invité

Scientific Production

Journal Papers

I authored or co-authored the journal publications that follow:

1. Hakim Mitiche, Dalila Boughaci, and Maria Gini. Iterated local search for time-extended multi-robot task allocation with spatio-temporal and capacity constraints. Journal of Intelligent Systems, (2):347–360, 2019. ISSN online: 2191-026X, Received: 2018-06-19, Published Online: 2018-12-21, Published in Print: 2019-04-24. Available online at https://doi.org/10.1515/jisys-2018-0267.

    - Submitted: June 19^{th}, 2018
    - First Review: September 10^{th}, 2018
    - Accepted: November 3^{rd}, 2018
    - Published online: December 21^{st}, 2018
    - Published in print: April 24^{th}, 2019
    - Weblink: https://www.degruyter.com/view/j/jisys.ahead-of-print/jisys-2018-0267/jisys-2018-0267.xml
    - Citations per document (4 years, 2017): 0.982, SJR: 0.193, SNIP: 0.481, CiteScore: 0.96
    - Index: DBLP Computer Science Bibliography, SCOPUS, SCImago (SJR), Web of Science - Emerging Sources Citation Index, ProQuest, Ulrich's Periodicals Directory/ulrichsweb, Google Scholar, ...
    - Home Web page: https://www.degruyter.com/view/j/jisys

2. Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. Robotics and Autonomous Systems, (90):55–70, 2017. Special Issue on New Research Frontiers for Intelligent Autonomous Systems.

    - Submitted: May 2015
    - First Review: March 2016
    - Accepted: October, 4^{th} 2016
    - Published online: October, 20^{th} 2016
    - Weblink: http://www.sciencedirect.com/science/article/pii/S0921889016306157

- DOI: https://doi.org/10.1016/j.robot.2016.10.008
- Citations: 49 (October, 7^{th} 2019)
- IF: 2.638, H index: 94, SJR: 0.711, IF (5 année): 2.809, SNIP: 1.838
- Home web Page: https://www.journals.elsevier.com/robotics-and-autonomous-systems/

Communications Internationales

Below are the international conferences I participated to and which I am the paper's author:

1. H. Mitiche, D. Boughaci, M. Gini, Efficient heuristics for a time-extended multi-robot task allocation problem, in: International Conference on New Technologies of Information and Telecommunication (NTIC), Mila, Algeria, 2015. http://www.centre-univ-mila.dz/ntic15/.
   DOI: https://doi.org/10.1109/NTIC.2015.7368756.
   Citations: 8

2. H. Mitiche, J. Godoy and M. Gini, On the tuning and evaluation of iterated local search, in: Proc. of Metaheuristics International Conference (MIC), Agadir, Morocco, 2015. www.lifl.fr/MIC2015/
   Citations: 3

3. H. Mitiche and D. Boughaci, Combinatorial Auctions for Task Allocation in Disaster Response, In proc. of the International Conference on Metaheuristics and Nature-inspired Computing (META), Sousse, Tunisia, 2012. www.lifl.fr/META2012/.

Others

I participated weekly to Pr. Gini's research team presentations. I presented the following poster during the department's annual workshop:
H. Mitiche and D. Boughaci. Task Allocation with Spatial and Temporal Constraints, University of Minnesota Computer Science and Engineering Department Open House. Minnesota, USA, 2013. Program guide: https://www.cs.umn.edu/sites/cs.umn.edu/files/open_house_program_2013.pdf

Acknowledgement

To Allah first and foremost, I acknowledge my existence and anything good I brought to life, including this thesis. There are many people that have earned my gratitude for contributing to my evolution during the post-graduate period. I particularly acknowledge to Pr. Dalila Boughaci the acceptance to supervise my thesis and supporting me during many years. I am thankful to the thesis jury for reading and judging my work. I am very thankful to Pr. Maria Gini for significantly helpling, motivating and inspiring me (as a researcher and a person). I don't think this work would have come to the present quality without Maria's research team help (back then at the University of Minnesota). So I am as much thankful to them for: giving weekly research work presentations, reading my papers, attending my presentations and discussing my work, co-authoring my papers or advising me on how to do research, directly or indirectly. In particular I mention: Dr. Ernesto Nunes, Dr. Julio Godoy, Dr. William Groves, Dr. Marie D. Manner and Dr. Mohammed El-idrissi. I do not forget the reviewers, even those who rejected my papers gently and constructively. To you, I am grateful for the critics and suggestions without which I couldn't grow up as a researcher and move forward to get here. Since I m forgetful and can't list all the people that I should acknowledge, now, I rather fairly settle to say that I acknowledge the help of any person or organization that contributed, somehow, to the conclusion of this work.

Dedication

To Allah first and foremost. To those who held me up over the years. To my parents and relatives. I dedicate this work also to Pr. Doumenji (my elder ante's husband) for heartedly motivating me to study and research every single time I meet with him. To my friend and colleague Hichem Benyagoub, who genuinely inspired and advised me, and sincerely, for repeatedly reminding me that to do what I like, what makes me happy, is more important then academic research. Finally, I thank my enemies and to some of my "friends" who somehow pushed me and spared me much time to work on this thesis instead of hanging out with them.

Abstract

The allocation of tasks with constraints on when, where, and in which order they need to be handled, by embodied agents, is an important research issue. Practical applications, to name but a few, include: warehouse automation, goods (or persons) pickup and delivery, surveillance at regular intervals at an enemy border, exploration in hostile environments (e.g., Mars planet exploration mission with NASA's geologist rovers) and urban search and rescue (e.g., in the aftermath of an earthquake). In this thesis, we approach the problem of allocating tasks to multiple physical agents – such as robots – mainly with metaheuristics. Specifically, we address contexts where tasks have temporal and workload constraints, while the (embodied) agents have capacity constraints and need to travel to execute the tasks. Furthermore, we assume the number of tasks is relatively large compared to the number of agents. The disproportion between agents and tasks, in addition to the problem constraints, preclude the allocation of all tasks. Consequently, we want to find the allocation that maximizes the number of tasks that the agents could handle by deadlines. Our task allocation problem is relevant to many applications that seek to automate missions by teams of robots, or to coordinate teams of humans (such as ambulances, firefighters and police during a natural disaster response). The main contribution of this research work is the development of an easy to implement metaheuristic, an iterated local search, which proved, experimentally, to be effective and efficient in allocating tasks to multiple agents. Our iterated local search is suitable for time-critical situation; it generally returns solutions of better quality than the baseline's and promptly, even when we interrupt the search at different cutoff times. Other contributions are the formulation of a new practical optimization problem, the study of the problem's complexity and how it relates to multi-robot task allocation, as well as, routing and scheduling literatures. Besides, we investigate the approach of task allocation through combinatorial auctions using a metaheuristic for auction clearing. We study or improve some greedy (fast) heuristics which are handy when the computation resource is limited or when prompter response times are required. We suggest to embed the proposed heuristics into sophisticated approaches – such as metaheuristics or hyper-heuristics – to achieve better results. Finally, we propose an approach tailored to evaluate and tune iterated local search that is biased toward improving the guidance of search.

Keywords: multi-robot task allocation, iterated local search, routing and scheduling, heuristics, metaheuristics.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACO	Ant Colony Optimization
BGP	Bid Generation Problem
CA	Combinatorial Auctions
CO	Combinatorial Optimization
EDF	Earliest Deadline First
enhILS	Enhanced Iterated Local Search
faPIH	Fast Parallel Insertion Heuristic
ILS	Iterated Local Search
LA1	Earliest Completion task First, with a One Step Look-Ahead search
LS	Local Search
MRLS	Multi Restart Local Search
MRTA	Multi-Robot Task Allocation
MRTA-STC	Multi-Robot Task Allocation with Spatial, Temporal and Capacity constraints
NTF	Nearest Task First
OP	Orienteering Problem
PIH	Parallel Insertion Heuristic
rILS	Randomized Iterated Local Search
SA	Simulated Annealing
SIH	Sequential Insertion Heuristic
TOPTW	Team Orienteering Problem with Time Windows
TS	Tabu Search
TSP	Travelling Salesman Problem
UAV	Unmanned Aerial Vehicles
USAR	Urban Search and Rescue
VRPTW	Vehicle Routing Problem with Time Windows
WDP	Winner Determination Problem
xPIH	Parallel Insertion Heuristic with Task Selection

Introduction

Perhaps throughout all history until the 20^{th} century, humans never thought or dreamed to be served or helped, except by animals or fellow humans. By the invention of computer machines and programs, last century, and particularly by the birth of Artificial Intelligence (AI), around 1950, scientists anticipated machines that smartly assist humans, substitute them, invade their daily life and even takeover over it (just to mention the extreme Science Fiction vision). Thus, in a few decades, AI brought many dreams to reality: a driver-less car that can safely drive in a crowded military base, an automatic translator that is practical, geologist rovers that operate semi-autonomously millions of kilometers away (NASA Mars mission) [1], a robot that rivalizes with a chess or ping pong champion, or merely a vacuum robot that cleans dust at home while you are busy away.

AI naturally evolved to distributed AI, to deal with complex problems where intelligence is split among many autonomous machines that cooperate or compete with each others. Distributed AI brought revolutionary applications along with challenging research issues that are often multi-disciplinary. Task allocation to multiple autonomous robots is one major challenge for distributed IA. The problem is concerned with which robot should execute which task when there are a multitude of tasks, robots and constraints such time, utility or capacity. It underlies many practical applications in robotic systems, such as exploration and surveillance missions using drones or emergency response coordination of robots or human rescuers [2, 3, 4].

In this thesis, we study task allocation to multiple physical agents, also known as multi-robot task allocation problem. In particular, we are interested in problems where the tasks have spatial, temporal and workload constraints, while the robots have a work capacity constraint and their number is limited, relatively to the number of tasks, to an extent that may preclude the allocation of all tasks— in conjunction with the other problem constraints. Given the inherent complexity of the multi-robot task allocation problem, we investigate heuristic solutions, particularly, metaheuristics, so that to return suboptimal, yet good quality solutions in a timely manner. Our main contribution is the proposition of an "iterated local search" that has a trade-off between solution quality and response time such that to be relevant in real-time applications, as emergency response (the application that initially motivated this work). Furthermore, we proposed a tuning approach for iterated local search that focuses on the search guidance modules. To offer prompt solutions to resource constrained contexts and a building block to meta-heuristics, we studied basic heuristics on the problem, through replicating, improving or proposing a new heuristic. Last but not least: (i) we showed why combinatorial auctions are not convenient for our problem; (ii) we studied the complexity of the proposed problem and (iii) we related the

problem to the relevant literature.

Our thesis dissertation is organized in the following chapters:

- Chapter 1, the first part of our background study, overviews the topics: combinatorial optimization, problem computational complexity, algorithm complexity and exact vs. non exact algorithms, particularly metaheuristics, that copes with complex problems.
- Chapter 2, the second part of the background study, broadly presents the problem that concerns our thesis, namely task allocation to multiple physical agents and the major approaches to the problem.
- Chapter 3 formally states the thesis problem, a multi-robot task allocation problem, illustrates it, studies the complexity and relates the problem to the relevant literature.
- Chapter 4 presents a market-based approach we investigated to solve our multi-robot task allocation problem. Namely, we describe a combinatorial auctions formulation of the task allocation problem, then we propose and evaluate some metaheuristic, a stochastic local search, to clear the auction (so that to allocate the tasks).
- Chapter 5 proposes a set of basic heuristics, which are greedy in nature, to approach the problem when the computation resource is limited or a prompt response time is required. Otherwise, these same heuristics can be embedded in advanced approaches, such as metaheuristics or hyper-heuristics, when more resources or response time can be afforded to obtain better solutions.
- Chapter 6 proposes an approach to tune and evaluate the metaheuristic iterated local search. We illustrate the approach on some iterated local search that we adapt the implementation to our problem. Through an experimental approach, we show how to check if the search guidance modules need tuning, so to improve the metaheuristic.
- Chapter 7 presents our main contribution: an improved iterated local search, that experimentally proved to be effective and efficient to allocate tasks with spatial, temporal and workload constraints, to multiple physical agents with limited work capacities.
- A conclusion of the thesis that summarizes the work we have conducted and sets some research directions that should be interesting to pursue in the future.

Chapter 1

Combinatorial Optimization and Metaheuristics

> Perfection has been attained not when nothing remains to be added, but when nothing remains to be taken away.
>
> Antoine de Saint-Exupéry,
> Pilot & Writer

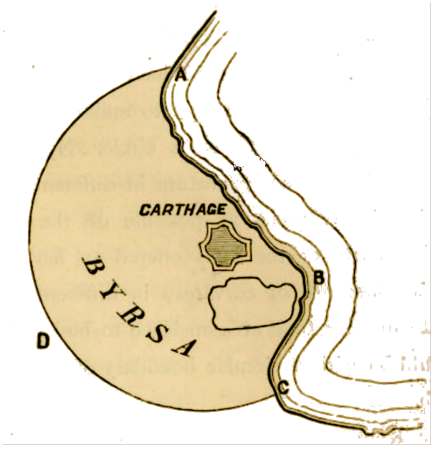
We provide basic knowledge on combinatorial optimization and metaheuristics. As many combinatorial optimization problems are difficult to solve, they are often approached with sub-optimal algorithms, such as heuristics or metaheuristics. Some multi-robot task allocation problems, as ours (Chapter 3), can be modelled as combinatorial optimization problems. Unfortunately, they are often hard to solve (cf., Chapter 2). The chapter is organized as follows. In Section 1.1, we give philosophical and historical perspectives on optimization. In Section 1.2, we formally define, illustrate and discuss possible variants of the optimization problem. In Section 1.3, we formally talk about combinatorial optimization where we present some well-known problems. In Section 1.4 and 1.5, respectively, we discuss the computational complexity of Algorithms and that of problems. Section 1.6 compares exact Algorithms with approximate or heuristic Algorithms. A brief overview of major established metaheuristics is given in Section 1.7. We briefly talk about the design and the evaluation of metaheuristics in Section 1.8 and we conclude in Section 1.9.

1.1 Philosophy and History of Optimization

Optimization is ubiquitous in nature and in everyday humans life. Men ever travel along the most convenient path, whether it be the shortest, the safest, the one with the highest availability of water or temporary shelter. Similarly, birds migrate through flyways that combine safety, food sources and a short distance. To optimize means to find, or select among given possibilities, the best one ("optimal") with regard to some criteria, such as efficiency, economy or appearance. The human being, by the endowed faculty of the intellect, has (more or less) explicitly optimized, since the dawn of times: which land should I farm or live in? which woman should I marry? which animal should I hunt or raise? which one

is the most suitable to ride and which one is the best to carry my goods?[1] Early historical accounts of optimization are about the isoperimetric problems. A clever Phoenician queen by the name of Dido [2] is the first recorded to have practically solved the basic isoperimetric problem around 900 B.C.

Dido's problem



Dido's solution on the Byrsa map

Figure 1.1: Queen Dido instructing her servants on how to lay out the rope made of a bull's strips (left side) to acquire the maximum of land (right side)

.

Dido fled her home city, Tyre, to escape from Pygmalion, her tyrant brother, after he murdered her rich uncle (and husband), Acerbas, and plotted to defraud her of the money. Once she landed in a gulf in North Africa (in today's Tunisia), Dido might have convinced the Berbers' chief, Hiarbas, to grant her as much land as she could enclose within the hide of a bull. She cut the hide into thin strips, joined them into a rope that she laid out in a semi-circle, using the sea coast as the complementary boundary (Figure 1.1). Thus, Dido acquired the widest area of land possible – on which she built Carthage – while satisfying the perimeter constraint. Interestingly, she perhaps implicitly suggested that among all shapes with equal perimeters, the circle is the one with the largest area.

A similar problem is reported some three or four hundred years later. Horatius Cocles, after saving his country, was granted, by Romans, as much land as he could plough round in a day. Cocles' problem, even in its simple version [3] is more complex than Dido's since the circle's radius must be estimated upfront such that its area could be enclosed by a deadline. In about 300 B.C., Euclid somehow proved that the circle has an area a larger than the area of a square or a rectangle's with the same perimeter. Zenodorus (200–140 B.C) and Ptolemy (90–168 A.D) wrote a treatise on isoperimetric figures. Later on (8^{th}–13^{th} century), Al-Kindi, Al-Hasan Ibn Al-Haytham and Abu Ja'far al-Khazin wrote commentaries or extended the work of the Greeks on perimetric problems such as Ptolemy's.

During the enlightenment era, the mathematical study of some optimization problems began to take off with the advent of calculus. Scientists like Newton, Leibniz and Bernoulli brothers, developed systematic ways to find an optimized path or shape of curve among

[1] Contrary to animals, humans often optimize deliberately, such as when selecting the cloths to wear (according to the occasion or the weather). Non deliberate optimization happens in the internal organs, as the control of the amount of enzymes to secrete to digest food.

[2] The basic isoperimetric problem was named after her as, Dido's problem

[3] Horatius might have considered that land vary in value and difficulty to plough.

some class of curves [5]. Issac Newton (1642–1729) studied the shape of the projectile that would have the least air resistance—so it can be exploited to throw missiles the farthest distance. In 1773, Lagrange formulated the problem of the shape of the strongest column (civil engineering). A related problem, which is the shape that gives the greatest torsional rigidity, was solved by George Pólya in 1948. The problem of how to lay down a railway with minimum cost between two cities emerged and was somehow solved, which enabled an efficient railway network that was behind the rapid growth of the US industry.

Some scholars, fascinated by nature, were convinced that this optimizes. George Pólya (1887–1985), conjunctured that the sphere is the 3-D shape with least heat loss; he gaves it as an explanation of why a cat curls into a ball during a cold winter's night. The most notable example of optimization in nature is the honeycombs of bees. Many writers, since the antiquity speculated, on why honeycombs are optimally built. When Thomas Hales proved, in 2001, that regular hexagons provides a least perimeter way to partition the plane into unit areas, it was the longest standing open problem in Mathematics. Another example is the least-time principal of Pierre de Fermat (1601–1665) that asserts that the actual path between two points taken by a beam of light is the one that can be traveled in the least time. In 1710, in his philosophical work Théodicé, Leibniz claims that the universe is the best possible. Such a philosophical claim had a great influence on the development of the principal of least-action, which is about the nature of motion and which was useful to modern Physics topics, such as general relativity theory and Quantum field theory.

Nowadays, optimization is prevalent in engineering, leveraged by technological advances and a competitive international market that values continuous innovation (such as engineering smart-phones with longer battery autonomy). Currently, optimization problems in industry range from trivial, such as the design of a smart phone with the most appealing look, to complex, such as the design of an airplane that is fast and fuel savvy.

1.2 The Optimization Problem

Formally, an optimization problem instance is a pair (S, f), where S is the set of possible solutions (possibly infinite but countable), ‘ or the search space, and $f : S \to \mathbb{R}$ is a function that associates a cost to each solution. The goal is then to find a solution $s^* \in S$ such that:

$$f(s^*) \leq f(s), \quad \forall s \in S \tag{1.1}$$

That search space's element with the best cost is called an optimum [4]. Since s^* minimizes f, we can specifically designate s^* as a minimum. The function f is often referred to as the objective function [5]. From now on, we adopt the following common abbreviation of Eq. 1.1:

$$f(s^*) = \arg\min_{s \in S} \ f(s), \tag{1.2}$$

[4] If many, they are called optima, or specifically *global* optima, contrary to *local optima* which are the best only in some search space regions

[5] In the case of a minimization problem – as above – the function f is called the *cost* or the *energy* function. In the case of a maximization, f is called a *reward* or *utility* function.

where "arg min" is an operator that finds an element of S which minimizes f [6]. It is customary (and correct) to formulate any optimization problem as a minimization. Given an objective function to maximize, say g (i.e., find $s^* \in S$, such that $g(s^*) \geq g(s), \forall s \in S$), we may put $f = -g$ and easily find an optimal solution (a maximum) by minimizing f.

Optimization problems can be categorized according to their characteristics. When the objective function is subject to constraints, it is a constrained optimization. If there is one objective function, we speak about a mono-objective optimization; otherwise, we are dealing with a multi-objective optimization. In the latter, we rather look for a solution that is not dominated at least on one objective (pareto-optimal). If the objective function and the problem constraints are linear (and continue), we usually do linear programming (e.g., the simplex method developed in 1947 by George B. Dantzig) to exactly solve the problem.

Unfortunately, many objective functions are not linear; thus they should be approached by not exact techniques, such as those based on mathematical concepts (e.g. the gradient descent) under some assumptions [6]. Namely, if the problem's objective function f and the variables x are continue, if we can calculate f's first derivate (f') and if f is unimodal (convex), we optimize f, when unidimensional, by solving the equation $f'(x) = 0$. Independently of the number of modes of f, we can use the gradient descent method (in case of a minimization). However, it is not globally optimal when f is multi-modal (i.e., many modes and minima), for the search can easily get stuck at one local optimum. Another gradient descent's drawback is that the search may easily stagnate at a saddle point [7] during the descent or even at a local maximum, since the objective function derivate there is null [6]. Furthermore, gradient descent efficiency depends on carefully setting the descent coefficient parameter α and the optimal solution is usually found within a certain range of precision. Gradient descent with restarts may instead be used to overcome the local optimality issue. Still, we need to tune carefully α parameter and run the algorithm long enough to increase the chance of finding a global optimum [6]. If we can compute the second derivative, the Newton method is preferred since it can avoid saddle points and it converges, relatively, prompter. Still, the Newton method does local optimization. Nonetheless, again we can overcome a local optimum by repeatedly restarting the search. In many cases, the objective function's derivatives are unknown. Consequently, we resort to other sort of optimization methods, such as heuristics, that are tailored to the problem at hand, or metaheuristics, that can fit to any optimization problem.

1.3 Combinatorial Optimization

A combinatorial optimization (CO) problem is characterized by a finite and countable set of solutions, S, which is usually a subset of all combinations or permutations of the solution components. Specifically, it is the problem constraints that reduces S, as in the following illustration. A foreign tourist wants to visit some cities of Algeria. He landed in Algiers and has to return there to fly back home. Since the tourist wants to spend as much time as possible visiting, rather than hitting the road; he needs to know the shortest tour that

[6] The function f may admit many optima; which one is obtained is a matter of indifference. By assuming that the set S is closed and not empty, we know that at least one minimum exists

[7] a saddle point is an optimum at which the first derivative of f is null ($f'(x) = 0$)

passes by all the cities exactly once, starting from and finishing at Algiers [8]. The problem can be formulated with a non oriented graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$ a set of n ($n \geq 3$) nodes (the cities), and $E = \{e_{i,j} | i \neq j$ and there is a path linking node i to $j\}$ the set of edges (direct paths between the cities). Let an arc $e_{i,j}$ corresponds to the shortest highway path between city i and j. The distances between nodes are given by the symmetric matrix:

$$D = [d_{i,j}]_{n \times n},$$

Where $d_{i,j}$ is the distance between nodes v_i and v_j and $d_{i,j} = d_{j,i}$ for each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$ [9]. There are $n!$ possible tours, with different length, that pass exactly once by each city. The solution set S is thus the permutations (i.e., combinations without repetition) of the n cities where we add the first element to each permutation. For instance, if $V = \{v_1, v_2, v_3\}$ then $S = \{\langle v_1, v_2, v_3, v_1\rangle, \langle v_1, v_3, v_2, v_1\rangle, \langle v_2, v_3, v_1, v_2\rangle, \langle v_2, v_1, v_3, v_2\rangle, \langle v_3, v_1, v_2, v_3\rangle, \langle v_3, v_2, v_1, v_3\rangle\}$, though some solutions are equivalent (e.g., $\langle v_3, v_2, v_1, v_3\rangle$ and $\langle v_2, v_1, v_3, v_2\rangle$Generally, a tour (candidate solution) is noted as $x = \langle v_{x_1}, v_{x_2}, \ldots, v_{x_n}, v_{x_1}\rangle$, where $v_{x_i} \in V$ and $\{v_{x_1}, v_{x_2}, \ldots, v_{x_n}\} = \{v_1, v_2, \ldots, v_n\}$. That is, in tour x, our tourist visits city v_{x_1} first, then v_{x_2} and so on until v_{x_n}, then he returns to v_{x_1}. The cost of a tour $x \in S$, $f(x)$, is x's length (Eq. 1.3), the goal is to find an shortest tour x^* (Eq. 1.4) and the problem is known as the (symmetric) traveling salesman problem (TSP) [7].

$$f(x) = \sum_{i=1}^{n-1} d_{x_i, x_{i+1}} + d_{x_n, x_1} \tag{1.3}$$

$$x^* = \arg\min_{x \in S} f(x) \tag{1.4}$$

To solve a CO problem instance, we may need to construct the solutions (e.g., tours for TSP) from their components (e.g., cities) by satisfying the problem constraints (e.g., to visit all the cities exactly once and in a tour). We should be able to measure the cost (e.g., tour distance) or value of a solution. A problem instance difficulty is affected by the problem size, such the number of cities in TSP, and by the instantiations of variables, such as the the number of links between the cities.

1.3.1 Some Well-known Combinatorial Optimization Problems

The following CO problems are practical to solve or to use as a testbed to evaluate or compare optimization techniques.

Maximum Satisfiability

We are given a set $U = \{u_1, u_2, \ldots, u_m\}$ of boolean variables and a set $C = \{c_1, c_2, \ldots, c_n\}$ of clauses. Let the function $\phi : U \to \{0, 1\}$ be a truth assignment for U. We say that $u \in U$ is true under ϕ, if $\phi(u) = 1$; otherwise (if $\phi(u) = 0$), we say that u is false under ϕ. If $u \in U$ is a variable, u and $\bar{u}$ are literals over U: the literal u is true if and only if $\bar{u}$ is false. A clause $c \in C$ is the disjunction of some subset of literals over U. Therefore

[8] He may might rent a car and would like to save on gas as well.

[9] We set $d_{i,j} = \infty$ when no direct path links v_i to v_j, to exclude non-existent paths from the solution.

c is satisfied by a truth assignment ϕ, if and only if, at least one of c's members is true under ϕ. For instance, let $U = \{u_1, u_2, u_3\}$ and $C = \{u_1 \vee \bar{u_2}, \bar{u_3}\}$. The first clause is true if and only if u_1 is true or u_2 is false. The problem (noted MAX-SAT) aim to find a truth assignment, ϕ, that minimizes the number of unsatisfied clauses. The assignment ϕ such that $\phi(u_1) = \phi(u_2) = 1, \phi(u_3) = 0$ is optimal since it satisfies all C's clauses (in the example). There are $2^{|U|}$ possible truth assignments. Similarly to the TSP, the MAX-SAT is NP-hard and used as a testbed for optimization techniques.

Quadratic Assignment

The problem is about efficiently assigning n facilities to n locations. Two matrices are given, A and B, where a_{vw} is the distance between locations v and w, and b_{vw} is the flow or weight between facilities v and w. The goal is to find an optimal assignment, which is a permutation $s = \{s_1, s_2, \ldots, s_n\}$ that minimizes the cost function:

$$f(s) = \sum_{v=1}^{n} \sum_{w=1}^{n} b_{vw} a_{s_v s_w} c_{vw},$$

where s_v denotes facility v's assigned location, $b_{vw} a_{s_v s_w}$ represents the cost of simultaneously assigning facility v to location s_v and facility w to location s_w and c_{vw} the distance cost. The problem is fundamental in transportation and facilities location, such as planning the placement of related factories, where the flow is the supplies amount to transport between the factories. The cost function encourages factories with high flows to be placed close together. The quadratic assignment, TSP and the timetabling are the hardest and most studied problems in operation research and optimization [8].

Timetabling

The timetabling problem, generally, aims to schedule some activities for some persons within a limited number of facilities (e.g., rooms) and during some time-slots (e.g., business hours). The university-course timetabling is a recurrent variant that features the sets of classrooms, time-slots, attending students, faculties and classroom requirements (e.g., video-projector, minimum attendance capacity) that are associated to the lessons. Feasible solutions are the assignments lesson/classroom/time-slot that satisfy the (hard) constraints: (i) no student should have overlapping courses, (ii) no faculty should have overlapping sessions and (iii) no lesson should be held in a classroom that cannot hold the students expected to attend. The objective is to further best satisfies some soft constraints (preferences), such as to minimize the occurrence of the events: (i) some student has only one hour of class in a day; (ii) a student has more than two hours of class in a row and (iii) a student has class in the last time-slot of a day.

1.4 On the Complexity of Algorithms

We inherently make algorithms efficient, particularly so when they may encounter difficult problem instances and/or limited computation power. The algorithm's efficiency is measured in terms of computer resources the algorithm consumes: computation time, memory

and/or network bandwidth. The computation time (or computational complexity) is often the only considered [10]. It varies in practice depending on the computer platform on which it is coded and ran [11]. Thus we often estimate the computation time in a uniform way (theoretically) by counting the number of basic operations necessary on some abstract machine [12].

Optimization problems usually require efficient algorithms. An algorithm that often takes a very long time to solve a problem's instance is useless to the user. An algorithm is deemed efficient for a CO problem, if it solves all the problem's instances within a reasonable computation time. The research and engineering community agreed upon that a computation time is reasonable if it grows as a polynom of the algorithm's input data size (variable). In the symmetric TSP (Section 1.3), for example, we can encode the number of cities with an integer and the distances (between cities) with a map of integers (that associates a distance to each pair of cities). Then, the size of a TSP instance is the sum of the amount of data in the map, which grows quadratically in the number of cities—thus, it is polynomial. The number of cities can be taken as a representative measure of the size of a problem instance; indeed, it is so in practice.

The computational complexity is often computed in the worst-case; when the input data make the algorithm consumes computation power the the most The worst-case complexity gives some guarantee on the maximum time the user has to wait before getting a response. It maybe then more practical to know than the best-case or the average-case complexities [9].

1.5 On the Complexity of Problems

So far, there is no efficient algorithm for a good number of CO problems (e.g., TSP, university-course timetabling and the quadratic assignment problem). They can be informally characterized as hard. Formally, we categorize computational problems according to their difficulty by analyzing the computational complexity.

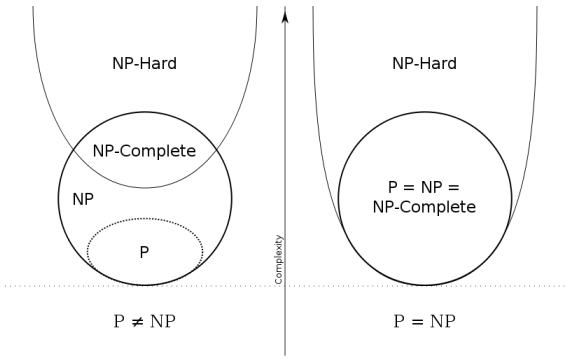


Figure 1.2: Euler diagram for P, NP, NP-complete, and NP-hard sets of problems.

[10] Storage capacity is growing large enough compared to computer programs needs, while the computation power is not catching up.

[11] In a broad sense, a computer platform is defined by the hardware performance (such as the CPU speed, the capacity of cache memory and temporary memory, the buses speed), the hardware architecture (number of CPU, structure of buses, temporary memory, etc)

[12] *e.g.* the Turing machine. We rather measure an *empirical* computational complexity when the algorithm is coded and ran on some actual machine.

1.5.1 Class P

It gathers problems with a known polynomial-time algorithm. A problem admits a polynomial-time algorithm if in the worst-case the computation time is $O(n^k)$, that is, upper bounded by some function of the form cn^k,w here n is the problem's input size, k is some finite integer constant as well as c. Class P's problem are easy to solve (tractable), such as searching a value in an array, sorting an a integers array or finding the shortest path between two nodes in a acyclic graph (If the plane is Euclidean, we get a P CO).

1.5.2 Class NP

Includes problems that, if we are given a "certificate" that a solution is correct [13], we can check the correctness in a polynomial time. For instance, the Hamiltonian-cycle problem is in NP. It aims to find a simple cycle that contains every vertex of a directed graph $G = (V, E)$ (V is the set of vertices and E is the set of edges). A certificate, for this problem, would be a sequence $\langle v_1, v_2, \ldots, v_n \rangle$ of n vertices ($n = |V|$) [10], which we can check the correctness in $O(n)$ time—which is, polynomial or more specifically linear, given that we have to check $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, n-1$ plus $(v_n, v_1) \in E$.

Still, we may not know an algorithm that can find the correct solution at first (for an NP problem). By convention, we say that a NP problem can be solved by a Nondeterministic machine in a Polynomial time (thus the acronym NP). Such a machine can be thought of as, fictitiously, capable of guessing a solution (to a yes-no question, for instance, in a decision problem). The time to guess a solution and prove its correctness is polynomial. We know at least that $P \subseteq NP$.

1.5.3 Class NP-complete

An NP problem is NP-complete, if it is at least as hard as any other problem in the NP class (Figure 1.2). An NP-complete problem is hard to solve or intractable. Most theoretical computing scientists think that NP-complete problems are intractable, for many such problems have been studied without finding a polynomial-time solution to any.

1.5.4 Class NP-hard

An NP-hard problem is at least as hard as any problem in NP (Figure 1.2), but it is not in NP. So, we cannot even check a given solution's optimality in a polynomial time.

1.5.5 Why we Study Complexity

The computational complexity informs the user on how to approach the problem. If some problem is proved to be in class P, the user already has an efficient algorithm for all its instances and may find a better one (for the problem is easy). If we find out that some problem is NP-complete, we have some evidence that it is intractable [14]. Then, we should better look for an approximate algorithm or a heuristic, unless we anticipate

[13] For a CO problem, a correct solution means an optimal one.

[14] Though all NP-complete problems seem to be intractable (so far), we cannot rule out that some NP-complete problem is tractable, as long as the question does $NP = P$ is open.

problem instances with small size or a manageable difficulty. If we rather look for an exact algorithm; chances are we won't find it or find one with a not practical computation time.

1.5.6 Complexity Proof

To prove that a problem is in P, it suffices to find an algorithm with a computation time that grows as a polynome of the input size. Generally, if we can reduce a problem A to another B in a polynomial time, we conclude that A is as hard as B. Then, to prove that a CO problem is NP-complete, it suffices to reduce it to some problem that is known to be NP-complete —for that purpose, many problems are proven NP-complete.

1.5.7 Decision *vs.* Optimization problems

The theory of computational complexity is developed for decision problems that aim to answer a "yes' or "no" question, but its findings can be extended to CO, for any CO problem is related to some decision problem [8]. Let Π_o be a CO problem and Π_d be a decision problem. Π_o is at least as hard as Π_d, if by solving Π_o we can solve Π_d. Let π_d be an instance of the decision TSP: given the cities and the distance between each pair; is there a tour that visits the cities once with a length that is less than some value l? Let π_o be the TSP instance that corresponds to π_d. If x_i^* is the length of the shortest tour $\in \pi_o$ (π_o's solution cost) then either $x_i^* < l$ or $x_i^* \geq l$. Either way, we can answer the question of π_d ("yes" and "no", respectively). Consequently, the TSP is at least as hard as the corresponding decision TSP.

1.6 Exact *vs* Approximate Algorithms *vs* Heuristics *vs* Metaheristics *vs* Local Search

Algorithms can be broadly categorized into either exact, approximate or heuristic. An exact algorithm guaranties to find an optimal solution. It is preferable if we can afford (or accept) its response time such as for class P problems. Otherwise, we look for an approximate algorithm or a heuristic. An approximate algorithm guaranties a bound near optimality, whereas a heuristic doesn't; nonetheless, it maybe easier to find. A heuristic [15] follows common sense rules that are peculiar to the problem at hand. Though it is a quick alternative, designing a "good" heuristic requires a good understanding for a new problem features. Metaheuristics [16] were introduced to enhance heuristics, by building on top of them, to offer a re-usable approach and reduce the development effort relatively to approximate algorithms. They are problem-independent high-level search strategies with modules to instantiate and tune for the problem at hand. Typically, a metaheuristic repeatedly interleaves two processes: (i) an incremental construction of a solution and (ii) a solution modification. It does not guaranty solution optimality.

Local search [17] finds solutions that are optimal only in some region of the search space (local optima). A solution neighbourhood needs to be defined: given a solution $s \in S$,

[15] The term "heuristic" has a Greek root that means: *I find or I discover.*

[16] The term, coined by Fred Glover in 1986, means "beyond I discover".

[17] Some authors abuse the use of the term when they refer, by it, to a metaheuristic that run a local search many times or to the heuristic that us embedded in a metaheuristic.

how to move to a "relatively" close-by solution $s' \in N(s)$, where $N(s)$ ($N(s) \subset S$) is s's neighbourhood [18]. A solution $\tilde{s}$ is a local optimum when no neighbour solution has a lower cost: $f(\tilde{s}) \leq f(s), \forall s \in N(\tilde{s})$. Technically speaking, a local search is an iterative procedure that searches, with some strategy, within the current solution's (s_i) neighbourhood $N(s_i)$ ($i \geq 1$), starting from a given solution s_0. When it finds a solution $s \in N(s_i)$ for which $f(s) < f(s_i)$, the procedure re-iterates with $s_{i+1} = s$. Otherwise, the search stops to returns s_i; the best-found locally solution (i.e., a local optimum). Again, a local search returns a solution that is the best in a small region of the cost function's landscape. In contrast, a metaheuristic, though not guaranteed to find a global optimum, explores many local regions. Consequently, it is very likely to find many solutions that are better then a solution obtained from a single run of the local search (which is embedded in the metaheuristic). A well-designed metaheuristic theoretically converges to a global optimum, when given infinite time and assuming that the cost function landscape is bounded [6].

A frequently used local search strategy is best-improvement. It enumerates all the solutions in the current neighbourhood to move to the one with the least cost (in a minimization problem) if this cost is lower than the current solution's cost. A local search with first-improvement policy moves to the first neighbour solution which cost is lower than the current solution's. To do so, it either deterministically enumerates the whole solution neighbourhood or just randomly sample from it, until an improvement is found or the whole neighbourhood is covered. The second alternative is appealing when the neighbourhood is very large. The initial solution s_0, that is seeded to a local search or a metaheuristic, can be generated randomly or by some heuristic if we prefer to start the search from a solution with quality. Random sampling of the initial or the neighbour solution makes the search stochastic. Then, many runs output different solutions with different costs [19].

1.7 Overview of Some Well Established Metaheuristics

Metaheuristics come in many flavours, often inspired by nature, such as by the foraging behaviours of ants, bees and bats. We briefly present some popular metaheuristics and give major applications thereof. Further metaheuristics and details can be found here [11].

1.7.1 Multi Restart Local Search

Local search (LS) returns an unsatisfactory solution, as it gets quickly trapped in a high-cost local minimum [20] We simply alleviate this drawback by restarting the search many times, through feeding arbitrary initial solutions. The search becomes a sequence of independent random samplings in S^*, where S^* is the set of local optima of S ($S^* \subset S$). Such a metaheuristic is known as random restart or multi-restart local search. The rational here is that if (i) the fraction of satisfactory local optima is sufficiently large and if (ii) these local optima are uniformly distributed on the search space, then by repeating the local search

[18] The definition of a neighbourhood is problem-dependent task left to the designer, which the local search performance depends, to a good extent, upon (with regard to solution optimality and cost

[19] Stochasticity, while useful, happens to be bothersome at the performance evaluation since we have to do many run replications and a statistical significance study which is not accurate

[20] We adopt the expression "low-cost" – *vs.* "high-cost" – to refer to local minimum which cost is sufficiently close to a global optimum cost (*satisfactory*).

procedure a sufficiently large number of times, the probability of hitting a satisfactory local optimum is then quite high. Unfortunately, often in practice neither of these assumptions is justified. Furthermore, as the problem instance's size increases, multi-restart local search gets lower and lower probability of finding satisfactory local optima [12]. Nonetheless, its triviality makes it a good performance yardstick (baseline) that any well-designed experimental analysis should at least include [8]. When confronted with a new problem, random restart enables a quick preliminary evaluation.

1.7.2 Iterated Local Search

Iterated local search (ILS) escapes local optima by resuming the search from a solution that is constructed from a partial alteration of the current local optimum. ILS should ideally always jump far enough from the current local optimum's region [21]; otherwise, the search remains trapped in the current local optimum. To be effective, ILS needs, as well, to not move too far away from the current local optimum region; otherwise the search may poorly explore – consequentely poorly exploit – the cost function landscape [22]. The hypothesis underlaying ILS is that low-cost local minima are clustered [8]. When so, appropriately adjusting the strength of the perturbation should enable the metaheuristic to effectively find the low-cost local minima. That depends also on how we perturb the current local optimum. To name but a few, ILS has been successful on the team orienteering problem, the quadratic assignment problem and the job shop scheduling problem [12].

1.7.3 Simulated Annealing

Simulated annealing (SA) is inspired by the annealing process of crystals. Based on statistical mechanics, a crystal is heated then cooled slowly, to obtain a perfect one [13]. SA for CO is a sort of random walk through the solutions space S. At each iteration, SA probabilistically selects a solution s' which is a neighbour to the current one, s (i.e., $s' \in N(s)$). The probability to select s' depends on s'' cost, $f(s')$, on the current solution's cost $f(s)$, and on a temperature T (a parameter) that changes over time

$$P_{s,s'}(T) = \begin{cases} 1 & \text{if } f(s') < f(s), \\ \exp^{\frac{f(s)-f(s')}{T}} & \text{otherwise,} \end{cases} \tag{1.5}$$

When s' is accepted, the search carries on from s. Otherwise, another solution $s'' \in N(s)$ is sampled and so on. In the standard SA's implementation, temperature T is initialized with some high value then it is slowly decreased along the search according to a given schedule. Consequently, at the beginning, the probability of accepting non-improving solutions is high; then it decreases (Eq. 1.5). Likewise, SA gradually shifts the search's tendency from exploration to exploitation. By doing so, SA quickly escapes from the basin of attraction of high-cost local minima which it likely encounters in early stages [8].

[21] A local optimum region, or the basin of attraction of the local optimum [12], is a part of the cost function landscape where a gradient descent from any point leads to the local optimum.

[22] We exploit a local optimum, when we look for a better optimum in "relatively" nearby neighborhoods, whereas we explore the cost function landscape, when we move the search to unvisited neighborhoods.

1.7.4 Tabu Search

Tabu Search (TS), introduced by Fred Glover, is nowadays among the most cited metaheuristics [8]. Basically, TS adopts a tabu list to escape from known local minima. When a local minimum is encountered, TS allows non improving local moves (small solution modifications) in the LS. However, the tabu list prevents the reversal of previous moves and thus avoids cycling through seen solutions [14]. Such list is a first-in first-out queue that contains a number of previously visited solutions. The search starts from an initial solution s_0. At each iteration i, the tabu list, TL_i, holds the l solutions visited lastly $(s_i, s_{i-1}, \ldots, s_{i-l+1})$, where l is the list length. The next solution s_{i+1} is set such that $s_{i+1} \notin TL_i$ and $f(s_{i+1}) \leq f(s'), \forall s' \in N(s_i) \setminus TL_i$. As TS walks through the space of solutions, it keeps in a short memory some recently visited solutions (tagged as tabu), so that they won't be selected again for a while. TS is applied with success on TSP. It is improved with a longer term memory of frequency to reinforce attractive solution components in [14].

1.7.5 Evolutionary Computation

Evolutionary computation (EC) is a collection of metaheuristics that borrow heavily from population biology, genetics and evolution. Famous EC includes the genetic algorithm, evolution strategy and evolutionary programming. The former is mainly applied in CO. Previously seen metaheuristics are trajectory based: they maintain a single solution. In contrast, EC is population based: it maintains a set of candidate solutions (a population) along the search. A candidate solution (or an individual) is described by a chromosome which genes holds the solution components. The basic EC algorithm constructs an initial population, then iterates through three procedures. The first one assesses the fitness [23] of every individual of the population so the second one can breed a new population (children). The third procedure joins the parents and the children, in some fashion, to form the next-generation population and so and so forth [6]. The breeding step selects parents from the population, then tweak them to make children, usually through mutation or recombination. The join step, usually, either completely replaces the parents with their children, or keeps fit parents along with their children. Before mating, the parents maybe refined through a LS, to pass to future generations an improved genetic material (as in memetic algorithms).

1.7.6 Ant Colony Optimization

Ant colony optimization (ACO) is a metaheuristic that is inspired by the foraging behaviour in ants colonies [15]. Scientists found out that ants efficiently find the shortest path between the colony nest and the discovered food source, through laying out pheromone [24] along the paths that leads to the food, which is exploited as a positive feedback and reinforced by other ants, during the many trips required to find and transport food [15]. The ACO metaheuristic uses artificial ants which cooperatively and stochastically construct a solution. Specifically, artificial ants iteratively add solution components to a population The ACO may exploit heuristic information about the problem when available. The artificial pheromone trails

[23] An individual's fitness is its value. By improving an individual's fitness we solve a maximization problem. In a minimization problem, the fitness maybe set as inversely proportional to the cost.

[24] a chemical substance deposited and sensed by many ant species to communicate, as their other sensory faculties are either weak or absent

in ACO to reflects the search experience acquired by the artificial ants [15]. The ACO performance of can be improved with a local search to refine a solution found by an artificial ant, before the update of the pheromone trail. ACO is a very successful metaheuristic on academic and real world problems. It has the advantage of being inherently ready for dynamic problems, for the artificial ants model can adapt to the problem changes at runtime.

1.8 Design and Evaluation of Metaheuristics

Contrary to common practice, metaheuristics are usually designed and evaluated mainly through experiments. The engineering process may go through several rounds of design, implementation, tuning and evaluation.

1.8.1 On the Evaluation of Metaheuristics

In departure from usual algorithms, metaheuristics are experimentally evaluated, because of a lack of theoretical tools and the limit of their utility,.In academia, standard benchmarks are used, which are synthetic and/or made from real data. When no benchmark is available, the practitioner should elaborate adequate test instances. In industry, metaheuristics are experimentally evaluated on problem instances encountered in reality. If the metaheuristic at hand is stochastic in nature [25], many run replications need to be carried out, on every problem instance, to assess the performance. Afterward, a statistical study should be conducted to draw sound conclusions. The results can be compared against optimal solutions, when available; otherwise they are compared against best-known solutions—when affordabl, exact methods, such mixed integer programming, maybe used to find the optimal solutions. Either way, one may need to experimentally compare the candidate metaheuristic against the state-of-art approaches, with regard to the solution quality – since suboptimal results are expected – and to the computation time. The experimental comparison should be aware of the heterogeneity of the computer platforms, on which the competing metaheuristics have been ran. The evaluation is further complicated when we are rigorous on how the competing metaheuristics have been tuned: as we discuss it later, while a metaheuristic's performance is sensitive to its free parameters setting, parameters tuning may have not been performed to the same extent on the metaheuristics under comparison.

1.8.2 On the Design of Metaheuristics

A Metaheuristic design is well based on experiments [16, 17]. Hooker advocates experimental analysis of the role of each component [16]. He suggests to single out a component to compare two, otherwise identical, implementations of the same metaheuristic. This paradigm, known as fractional design, should inform the designer about the performance impact of the metaheuritic components (e.g, the LS of ACO). Based on this approach, Xu et al. analyzed the effectiveness of each components of a TS [17].

[25] In fact, the bulk of metaheuristics are stochastic, that is why they are sometimes referred to as *stochastic local search*. A stochastic metaheuristic's performance is a random variable.

1.8.3 On the Tuning of Metaheuristics

Metaheuristics come with some free parameters that need to be set before they becomes functional. A metaheuristic's performance depends to a good extent upon parameter tuning. For high quality results, a considerable effort and technique is needed The metaheuristic design can be also seen as a component tuning: among some possible instantiations of the modules, which combination yields the best performance? [8].

The tunning phase is often done by hand in a trial-and-error fashion, guided by some rules of thumb [12]. However, this practice shows limits and drawbacks in industry and research. For large-scale industrial applications, tuning metaheuristics so is time consuming, labor intensive and requires a skilful person [8]. In academic research, metaheuristics are often tuned by researchers who are not equally acquainted with the metaheuristics under analysis or, though acting with the best faith, researchers maybe more keen on tuning their metaheuristic, then on checking whether others' have been well tuned [8]. Consequently, it is likely that the experimental comparisons reported in academic research papers have conclusions undermined by how the tuning was carried out. Surprisingly, few researchers seem to be aware of such limits, and often their papers do not even elaborate on how the tuning was performed [8].

Another approach of tuning is to use a gradient descent (or a local search) in the space of free parameters values [18]. Though suboptimal, such an approach is systematic and can find good parameters settings. More elaborated systematic tuning approaches have been attempted, such as with an iterated local search [19]. However, another tuning problem with the metaheuristic used as tool —nonetheless, some quick by-hand tuning should yield acceptable results. A more recent and advanced systematic tuning method is based on machine learning. In that direction, online tuning adjust the free parameters values during the search. Tuning during runtime is often based on reinforcement learning. It is relevant when we do not know the distribution of the problem instances which the metaheuristic will encounter. Otherwise, an automated off-line tuning can be conducted based on racing algorithms, along with a statistical analysis to speed up the process [20]. For more insights, the reader is directed to Birattari's book [8], which is based on his PhD thesis, on automated metaheuristic tuning based on machine learning.

1.9 Conclusion

This background study was about combinatorial optimization and metaheuristics. We gave the historical and philosophical perspectives, we gave definition of optimization and combinatorial optimization and illustrated with some practical problems. We talked about the study of the computational complexity of problems and algorithms and showed the utility thereof for CO. We briefly described the kinds of optimization algorithms according to their principle and accuracy in finding a global optimum. We emphasized that metaheuristics, while not exact, are: re-usable (contrary to heuristics), can achieve high performance on complex CO problems (often the state-of-the-art one's) and are promptly available (compared to approximate algorithms). We gave an overview of recurrent metaheuristics and the applications where they proved successful. We concluded by talking on how metaheuristics are designed, tunned and evaluated.

Chapter 2

Task Allocation to Multiple Physical Agents

As a second part of the thesis' background study, we review the literature on task allocation to multiple physical agents. In Section 2.1, we gently introduce the problem through an illustration, a short definition and a listing of major applications. In Section 2.2, we talk about task allocation to robots. In Section 2.3, we expend on some major applications, in particular, emergency response, which initially motivated this research. Section 2.4 briefly review the analysis and classification of multi-robot task allocation problems. In particular, we present a new taxonomy by Nunes et al. [21], to which we contributed. Section 2.5 summarizes the main approaches to allocate tasks to physical agents. In Section 2.6, we discuss some routing and scheduling problems that are related to multi-robot task allocation. Section 2.7 expands on multi-robot task allocation to teams of robots

2.1 Introduction

2.1.1 Illustrative Example

Think of a shipping company that sells an item every hour. Instead of a human, a robot at the warehouse could receive the purchase order, fetch the item, pack it and prepare the package for a pick-up by a transporter. What happens when the company sells 20 items every hour? What about 20 items every minute? What about 20 items a second? Amazon, a large online shopping company, sold 5.5 million items on "Black Friday" [1], in 2014. With 64 items ordered per second that day, a single robot would be hard-pressed to keep up with the orders. An alternative substitute for the robot, on such a busy day, would be a large team of robots that cooperate effectively in handling the orders. Specifically, each shipping robot would have to plan an efficient route through the warehouse to fetch items without colliding with the other robots, without taking items that another robot is handling, eventually, while considering (in its route plan) requested items that are out-of-stock, but will be restored soon [21].

[1] a busy shopping day that is popular in North America.

2.1.2 Definition

Task allocation to multiple physical agents addresses the question of which agent should execute which tasks, so that some total cost is minimized and/or some total reward is maximized [22]. In the shipping problem above, the total reward maybe the time saved by the robots (the agents), compared to humans, to handle the purchase orders in a business day. The total cost maybe the sum of work durations, total travel distance in the warehouse or the global battery power consumption that is necessary for the robots to handle the ordered items. Typically, the physical agents are embodied autonomous entities which are spatially dispersed, may relocate or travel to task locations. They maybe: human teams, wireless sensors, transport vehicles, unmanned air vehicles, unmanned ground vehicles, etc.. Splitting tasks among a group of physical agents is required when a single agent cannot perform all the tasks alone. Otherwise, we may do so for a synergy benefit, such as to speedup task execution (as in [23]), to deliver a better execution quality (as in [24]) or to avoid a single point of failure (as in [25]).

2.2 Multi-Robot Task Allocation

Multi-robot task allocation (MRTA) is an important class of task allocation to multiple physical agents problems. It involves autonomous robots, such as rovers or unmanned air vehicles. The MRTA issue is at the core of multi-robot systems research. It started in earnst in the 90's, when researchers started pulling together teams of robots to accomplish mission that consists of multiple tasks. MRTA research has been growing in academia and industry along with the proliferation of autonomous robots use in civil and military domains. While studied mainly in AI, MRTA has theoretical grounding in areas as diverse as Mathematics, Operations Research, Computer Science and Robotics. Research topics related to MRTA include: the assignment problem, combinatorial optimization, scheduling and routing, distributed computing and distributed AI.

When we allocate tasks to autonomous robots, the objective is typically to minimize the travelled distance—since, usually, the robots' battery power is limited—or to minimize the mission's time, such as in time-critical missions. The allocation problem is basically constrained spatially and temporally. That is, the robots have to travel to the task's location and the task may only be available during a certain, potentially short, period of time. Advanced applications have further constraints on tasks, such as execution precedence or synchronization [21]. For instance, let us have tasks x, y and z. A precedence constraint maybe "x should always be executed before y", while a synchronization constraint maybe "y and z have to be executed simultaneously". The robots are generally constrained by their capabilities and resource availability to handle the tasks, such as the amount of work per time unit and battery power.

The research works nowadays consider different spatial and temporal constraints (on tasks and agents), advanced task utilities/costs, dynamic and fault-tolerant allocation, probabilistic and stochastic models to handle uncertainty. The approaches are as diverse as market-based planning, Markov decision processes, decentralized scheduling algorithms and distributed constraint optimization. Since task allocation to multiple physical agents is mainly studied under MRTA designation, we refer to it so from now on. Again, MRTA

involves, besides autonomous robots, human-operated vehicles, humans, etc.

2.3 Some Interesting Applications

Task allocation to multiple physical agents has many applications. Notable applications include robotic missions, such as mapping, exploration and surveillance in hostile environments (e.g., NASA's rovers sent to the planet Mars), multiple vehicles routing and scheduling (e.g., the delivery of goods [26]) and tourist trip planning [27]. The urban search and rescue [2] is another notable and emerging application.

2.3.1 Mapping, Surveillance and Exploration

Mapping, surveillance and exploration through autonomous multi-robot systems are particularly relevant when the environment is hostile to humans, such as an enemy border territory, a radiated zone or a deep sea. Such robotic missions underlay task allocation and task execution, usually, by unmanned ground or aerial vehicles. In the Centibots project, for instance, a hundred robots system was developed to map and explore indoor areas, in a dynamic and reliable fashion. The system's architecture anticipates dynamic teaming to perform tasks that robots cannot perform individually, such as tracking targets [25]. In the military domain, unmanned aerial vehicles (UAV) may have to be assigned to disparate zones in a war area, to efficiently inspect locations where possibly explosive devices are hidden [28]. In a civil context, UAVs may have to be effectively allocated to riots that arose during a sport event or a public demonstration, so to adequately direct the police forces. Task allocation to UAV should further satisfy constraints on energy consumption, communication ranges and some tasks synchronization, such as 3D monitoring of a forest fire [29]. In NASA's Mars exploration mission, twin geologist rovers, "Spirit" and "Opportunity", were sent to study the land and look for water on the unknown planet. The robots have to challenge an unknown environment, mostly autonomously, with limited energy power [1].

2.3.2 Emergency Response: Disaster Mitigation

Emergency response deals with important social issues that involve urgent and large scale handling, such as natural disaster mitigation [2], riots monitoring or law enforcement at a city scale [4]. Disaster mitigation is particularly getting more attention the last two decades, during which the world witnessed an increasing number of devastating and deadly disasters. Just to name but a few, we cite Fukushima earthquake and nuclear leak (2011), Gulf of Mexico oil spill (2010), the tsunami that hit South East Asia (2004) and the earthquake and flood that hit Algiers (2003 and 2001, respectively). Disaster mitigation requires the coordination of rescue operations, such as in the aftermath of an earthquake (the typical example). These include extinguishing fires, digging out victims under rubble, evacuating injured civilians to hospitals and clearing blocked roads [2]. The rescuers (e.g., firefighters, ambulances and police forces) may have a limited availability—as when the number of rescuers is small relatively to the number of emergencies—and their skills are complementary. For instance, paramedics can give first aids to casualties only after the firefighters put

out a fire, evacuate an injured or dig out a civilian under rubble. Often, a large number of rescue tasks show up after an important natural disaster. Consequently, rescuers need to be effectively assigned to rescue tasks. Besides heterogeneous capabilities, the rescuers may as have different performance capacities. For instance, some firefighters squad maybe more efficient than another to put out a fire (say because of better skills, more crew or the possession of better tools [3]). The emergencies may have different priorities (e.g., digging out an injured civilian is more important then putting out a fire in an empty building), maybe constrained by time (e.g., a building resists fire only f or a limited period) and have some difficulty (e.g., a fire requires a certain amount of work depending on the building's material, the volume and the period it has been burning).

Emergency response is a challenging application for MRTA: (i) the tasks have to be promptly allocated and possibly re-allocated; (ii) the number of agents is often very limited relatively to the number of tasks; (iii) the tasks may change or disappear; similarly, the agents may become unavailable or new ones may arrive and (iv) the communication may become limited as when the disaster impairs the radio infrastructure. Emergency response may include exploration, as searching for victims or after a natural disaster. When it so at a large scale (as a city), we usually referred to it as Urban Search and Rescue [2].

2.3.3 Emergency Response: City Law Enforcement

To enforce law in a city, police forces are allocated to routine patrolling and, meanwhile, they are re-allocated to respond – often in teams – to eventual incidents [4]. Each incident (task) has an importance that ranges from low (e.g., a noise complaint) to high (e.g., a murder) and a workload that indicates the amount of work necessary for clearing the incident. Multiple police officers may work together, by sharing the task's workload, especially on important tasks to improve the response quality, such as to dissolve a fight before it degenerates into physical or property damages and to arrive to the crime scene before the perpetrators escape. Thus, tasks in the city law enforcement problem are modelled with soft deadlines, so to allow late tasks handling. Yet, late handling is discouraged by decreasing the task utility over time.

2.3.4 People and Goods Pickup and Delivery

Picking up or dropping people off during a car tour is a sort of task allocation to multiple physical agents. Imagine, for instance, a radio taxi company with many taxis that are called continually by clients who want to be picked up at a certain location and dropped off at another. For an economic reason and for a better client satisfaction, the company has to efficiently allocate the taxis to the riders. Such a problem can be formulated as task allocation to multiple physical agents which are the taxis and the tasks are people pick up and drop off. The constraints concern time, vehicle routing and vehicle capacity (e.g., to drop off a client at an appointment on time; available routes, to save fuel or travel distance and the cab cannot carry more than four riders simultaneously). As to the objective, it maybe to service as many clients as possible (to increase benefit and availability to clients). The problem can rather be inherently modelled as an online task allocation, since the client requests come on the go.

Similarly, the goods pickup and delivery problem [30] implies task allocation to multiple physical agents. A typical example is mail pickup and delivery by a postal service company. The physical agents are human-operated vehicles, such as trucks of a mail service company. Interestingly, the advent of autonomous cars, such as the Google car equipped with AI or Amazon's parcel delivery autonomous drone [2], would enable in the future an automated transportation and task allocation. The pickup and delivery of mails (or parcels) have spatial, temporal as well as a load capacity constraints. Goods or people pickup and delivery problems have rather been notoriously studied as a routing and scheduling problems [31].

2.4 Classification and Analysis of MRTA problems

The multitude MRTA applications gave birth to multiple problems that differs on the assumptions made on the agents, the tasks and the allocation. Consequently, many approaches were devised to specifically address the assumptions peculiar to the problem at hand and cope with its difficulty. Analyzing and classifying MRTA problems, then identifying (or proposing) appropriate approaches is a research necessity, even more since MRTA problems are studied separately in different fields (such as, Robotics, Operation Research and Combinatorial Optimization). Knowing a MRTA problem's class enables to relate the problem to possibly well-studied problems. Consequently, the researcher may rapidly find the relevant state-of-the-art, be advised in the design of a new solution and be able to do a sound evaluation against the available competitors and benchmarks.

2.4.1 Gerkey et al.'s Taxonomy

Early works in MRTA were conducted in an ad hoc manner and validated empirically only. Motivated by the lack of a theoretical framework, Gerkey et al. attempted to formally analyze and classify existing MRTA problems [32]. Then, they proposed optimal algorithms, approximations or suggestions for the synthesis of new approaches which are grounded on theories in Operation Research and Combinatorial Optimization. Indeed, in the first attempt to analyze and classify MRTA problems [32], Gerkey et al. show how many MRTA problems are viewed as instances of well-studied optimization problems.

Gerkey et al.'s broadly categorize MRTA problems according to three axes that are concerned with major characteristics of the problem (respectively, the robots', the tasks' and the allocation time horizon's characteristics): (i) how many tasks can a robot perform simultaneously, (ii) does a task requires (or permits) multiple robots working simultaneously and (iii) can all the tasks be allocated upfront or as they appear. Specifically, an MRTA problem is classified as follow:

1. Single-task robots (ST) vs. multi-task robots (MT): ST robots can do at most one task at a time, while MT robots can work on multiple tasks simultaneously.

2. Single-robot tasks (SR) vs. multi-robot tasks (MR): SR tasks require exactly one robot in order to be completed, while multiple robots are needed to complete an MR task.

[2] https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011

3. Instantaneous (IA) vs. time-extended (TA) assignments: In IA, tasks are allocated as they arrive, while in TA, tasks are scheduled over a planning time horizon.

For example, task allocation in Ramchurn et al.'s work [23] differs from Nair et al.'s work [33], with respect to the second axis of the taxonomy. In the former, many agents can simultaneously work (as a team) on a same task (i.e., MR tasks), whereas in the latter, the agents perform each task individually (i.e., SR tasks). In [23], the presence of a number of tasks that is large relatively to the number of agents, in addition to tough task deadlines require that the robots work simultaneously on some tasks, so that: (i) to not miss urgent tasks and, assuming synergy in team work, (ii) to handle more tasks. In contrast, [33] assumes a simple MRTA problem where tasks maybe all allocated without team work. In both [23] and [33], the agents cannot perform more than one task simultaneously (i.e., ST robots). In [23], the authors assumes that the tasks are known upfront (i.e., TA allocation), whereas in [3], they consider a dynamic environment where some tasks cannot be known ahead since they appear during the allocation while others may disappear. If so, a TA allocation model should involve a re-allocation policy.

Gerkey et al.'s taxonomy is limited for assuming that tasks are independent and the sole concern for settings where the robots cooperation is intentional. Specifically, with regard to the first limitation, as pointed out by the same authors [32] and by [22], the taxonomy does not distinguish nor deal with problems where tasks have utilities that are interrelated (a limitation addressed later by another taxonomy [22]). Another limitation is to not care whether tasks have temporal or precedence constraints or not (addressed later by Nunes et al.'s taxonomy [21]).

2.4.2 Korsah et al.'s Taxonomy

Korsah et al. propose a taxonomy that adds levels under Gerkey et al.'s, to better specify some MRTA problems and to assess their difficulty. The taxonomy "provides a useful way of categorizing task allocation problems in a manner that is strongly related to problem difficulty" [22]. It is based on the following considerations: (i) does a task's utility solely depends on which agent is assigned to it, on the other task assignments of that same agent or on the task assignments of the other agents, as well; (ii) is the task simple, thus ready for allocation, or compounded such that to require decomposition (or multiple decomposition) and the allocation of sub-tasks. Based on tasks utilities interrelation, the taxonomy identifies four MRTA categories:

1. No Dependency (ND): where the effective utility of an agent for a task does not depend on any other task or agent in the system.

2. In-Schedule Dependency (ID): where the effective utility of an agent for a task depends on what other tasks that agent is performing (i.e., on the agent's schedule).

3. Cross-Schedule Dependencies (XD): where the effective utility of an agent for a task depends not only on its own schedule, but also on the schedule of other agents.

4. Complex Dependencies (CD): where the effective utility of an agent for a task depends on the schedule of other agents in the system in a manner that is determined by the particular decomposition of tasks that is chosen.

The taxonomy further sub-categorizes MRTA according to four types of tasks, depending on the task decomposability. Allocating decomposable tasks involve a task decomposition step before allocation.

1. Elemental task: an atomic task (a single action) that cannot be decomposed and can be performed by a single agent (i.e., SR task).

2. Simple task: either an elemental task or a task that maybe decomposed by one agent before being executed by that same agent (i.e., SR task).

3. Compounded task: task that can be decomposed into multiple tasks and executed by many agents, provided that there is one way of decomposition (i.e., MR task).

4. Complex task: a compound task that can be decomposed in several ways, at least one of them is feasible for allocation and where the subtasks can be allocated to multiple agents (i.e., MR task).

Korsah et al.'s taxonomy is particularly concerned with allocation problems where tasks are inter-related, because of dependent agent-task utilities or because of a composition relation among the tasks. As such, the authors narrow down Gerkey et al.'s MRTA classification. They provide examples, mathematical models and solution approaches for each MRTA class [22].

2.4.3 Nunes et al.'s Taxonomy

Nunes et al. recently (2017) extended Gerkey et al.'s MRTA taxonomy with a fourth classification axis that is concerned with time-extended MRTA problems only [21]. Specifically, the authors distinguish between allocation models where tasks have temporal constraints of the sort time windows (TW) and allocation models where tasks have precedence or synchronization constraints (SP). A time window constraint normally gives a time interval that restricts when a task can be executed. Usually, it specifies the task's earliest start time and latest finish time that are allowable —sometimes, a TW further constrains the task's latest start time and the earliest finish time, or just the latest finish time (i.e., a deadline). Precedence and synchronization constraints impose some complete, or partial, ordering of tasks executions —e.g., task x should be done before, after, or at the same time as task y. Nunes et al.'s taxonomy further narrows the classification down, by subcategorizing a MRTA problem according to [21]:

1. Hard vs. soft temporal constraints. Hard temporal constraints cannot be violated (e.g., surveillance in an enemy border or routing perishable goods), while soft temporal constraints allow violation with penalty (e.g., a robot vacuum cleaner that dusts the house before the householders return from work).

2. Deterministic vs. stochastic models of task arrival, task reward, robot travel time, etc. In deterministic models, the output is completely predictable by the initial conditions, while stochastic models assume some uncertainty on the allocation. The uncertainty is addressed with probability distributions, such as, the distribution of task arrival and the distribution of robot travel times.

Nunes et al.'s discuss some temporal models, in addition to time windows, that are useful to represent time constraints, such as simple temporal networks. Furthermore, the authors identify and illustrate common mono-objective optimizations in the MRTA literature, such as: minimize the sum of the robots path costs or the robot path cost with the maximal cost, maximize the profit, minimize the number of robots to use. The new taxonomy is more comprehensive as it takes into account some common execution dynamics. The paper [21] also discusses the algorithms, approximations and heuristics proposed to the problems in each MRTA class, by splitting them into centralized and decentralized. The work draws parallels between MRTA with temporal and ordering constraints and related problems, such as the assignment problem, VRPTW, Job Shop Scheduling and TOPTW. Throughout the paper, the authors discussed models and solutions coming from these areas.

2.5 Approaches to multi-robot task allocation problem

Approaches to MRTA can be categorized according to various criteria. With respect to decision making, we distinguish centralized approaches [34, 23, 33] from decentralized approaches [3, 35, 24, 36, 37]. Centralized approaches assume a (single) central decision maker responsible for assigning tasks to agents. In decentralized approaches, however, decision making is either performed through a decentralized algorithm, which execution is shared among the agents, such as decentralized constraint optimization [3, 35, 24, 38], or the agents independently decide which tasks to take. In the later case, task allocation emerges from simple and independent robots' behaviors, which are inspired by nature (eg., ant colonies, bee swarms, flocks of birds, etc) [37]. For instance, Krieger et al. demonstrated how a group of robots forages for some objects based on the division of labor in ant colonies [36]. Usually in a decentralized setting, agents have a limited awareness of the peer agents and/or of the tasks around. When approached with decentralized constraint optimization, task allocation to multiple robots problem is formulated as a constrained optimization problem, whereby tasks are allocated through a distributed algorithm ran by multiple robots [24, 38, 3]. Possibly, decentralized constraint optimization includes search pruning and/or communication reduction techniques for scalability [24, 38].

Decentralized task allocation is relevant in dynamic environments where it facilitates fault tolerance and adaptiveness to change (in tasks or agents). It is further relevant when the communication is constrained, in bandwidth and message size, or unreliable. In contrast, centralized task allocation assumes that the communication infrastructure is available and reliable. While it has single point of failure drawback, nonetheless, centralized task allocation is more effective when the environment is fully observable and static [3, 33].

Approaches to MRTA can further be described as cooperative when the agents explicitly collaborate to perform the tasks [24, 39, 40]. When the agents are self-interested, but might implicitly cooperate when they try to maximize their individual utility, then we talk about competitive task allocation [41]. A cooperative approach to task allocation can be centralized or decentralized; while a competitive approach is inherently decentralized. Further classification and discussion of MRTA solutions are given in [21]. Advanced MRTA problems consider execution failure or other contingencies such as the departure and/or the arrival of robots/tasks. Then, the MRTA problem is formulated such that to allow task

re-allocation and it is approached with strategies that are robust to change [42, 38].

2.6 Routing and Scheduling Problems

Other formulations related to task allocation to multiple physical agents are for problems concerned with the routing or scheduling of vehicles or persons, such as the vehicle routing problem with time windows (VRPTW) [26], the team orienteering problem with time windows (TOPTW) [43], and the multiple repairman problem with time windows [44]. In VRPTW, a set of nodes, with a certain load capacity demand, has to be visited by a number of vehicles, possibly with a total load capacity constraint. The vehicles need to start and finish at a depot node within a given time window [26]. The number of vehicles needs to be minimized. The VRPTW underlies multiple applications in transportation, such as the pick up and delivery of goods to clients by a shipping company. Similarly, the TOPTW [43] is concerned with visiting multiple nodes within given time windows. However, the nodes have to be selected, since they have different rewards and the vehicles may not be able to visit them all. Indeed, the objective is to construct non overlapping tours for the vehicles, such that to maximize the total collected reward. Contrary to VRPTW, the nodes do not require a load ability and the number of vehicles is usually fixed.

By driving a parallel with MRTA, VRPTW and VRPTW like problems take into account the temporal and spatial constraints on the agents and the load and delivery tasks; nonetheless, they differ in some ways. VRPTW is concerned with both the agent routing and tasks scheduling; which is not always the case in MRTA. The assumption of a load capacity on the agents is specific to some MRTA problems where there are objects to transport. The consideration of rewards upon task completion, as in TOPTW, is present in MRTA [24, 4]. However, in MRTA the number of agents is usually limited and may decrease due to robot's failure (e.g., because of a software bug, a mechanical issue or battery power exhaustion). Reward maximization in TOPTW is relevant to MRTA where tasks have utilities (or priorities) and the problem is over-constrained (i.e., it may not be possible to allocate all the tasks). Both VRPTW and TOPTW ignore task exploration and communication issues that are inherent to MRTA [21].

In workforce routing and scheduling problems, such as the multiple repairman problem [44], a company's crew needs to be routed to multiple sites and scheduled therein to perform some work. For instance, technicians of a Telecoms company who has different skills, tools, and spare parts, need to be assigned to daily maintenance requests, in a way that minimizes the clients waiting time [45]. Another example is the assignment of caregivers to provide health-care services to patients at home (e.g. the elderly or the disabled) [46]. Similarly to MRTA, home health-care may require staff with different qualifications (e.g., nurse, therapist, physician, social worker or food courier) and skills (e.g., spoken languages). Moreover, some health-care tasks require more than a single caregiver (e.g. lifting a disabled person) or have precedence constraints (e.g., a drug must be administered a certain time before serving a meal). Approaches to routing and scheduling problems are usually of an approximate or heuristic nature and they are usually computed off-line [31]. The interested reader is directed to the recent survey by Nunes et al. [21] for more optimization problems that bears similarity with MRTA.

2.7 Task Allocation through Coalition Formation

A task is allocated to an agents group, or a coalition, when it cannot be executed by a single agent, or when it is possible and more convenient that many agents work simultaneously on the task [47]. For example, consider again the case of the online shopping company when it operates with robots (Section 2.1.1). When a heavy item is ordered, two or more robots are required to transport the item in the warehouse, from the shelves, to the packing and picking up place and to load it into the carrier vehicle. Similarly, if we assume that some robots are idle (available), we should assign multiple robots to handle a new order if we aim to improve the item's delivery time. Furthermore, allocating coalitions of agents to tasks is particularly useful when there is a synergy in the team work that increases the system utility. Consider, for instance, an emergency response to many fires in a city. The firefighters if they work in coalitions may extinguish the same number of fires, but earlier then if they have worked individually (on each fire spot). Extinguishing fires rapidly not only mitigates the material damage, but it may also increase the number of handled rescue tasks, particularly in case the number of the tasks is larger then the number of the rescuers [23, 3].

Usually agents coalitions have different values (utilities). A coalition value is an estimation of the utility that is actually gained by forming the coalition. In online shopping, for instance, the coalition value of some buyers maybe the total discount earned when they buy a large quantity of some product, altogether. In our illustrative example, a coalition's value can be some number that is inversely proportional to the total time necessary for the coalition's robots to transport and pack the ordered item. The coalition formation problem requires first to compute the values of possible coalitions, before to search for the best coalition(s) to form. Let $A = \{a_1 \dots a_n\}$ be the set of agents, then C_A the set of coalitions that the agents may form gathers all subsets of A:

$$C_A = \{\{a_1\} \dots \{a_n\}, \{a_1, a_2\}, \{a_1, a_3\} \dots \{a_{n-1}, a_n\}, \dots \{a_1 \dots a_n\}\}$$

Let $u : C_A \rightarrow \mathbb{R}$ be the coalition value function which assigns a value to each coalition possible. When the coalition work is synergistic (i.e., the coalitions have an upper additive value) [23], we would have coalitions values of the kind $u(\{a_1, a_2\}) \geq u(\{a_1\}) + u(\{a_2\})$, where $a_1, a_2 \in A$. Finding the best coalitions to form consists in splitting C_A into disjoint subsets (coalitions), such that the total sum of the coalitions values is maximized. The problem known as the coalition structure generation [47] and is NP-complete, since it includes the Set Partitioning Problem [48]. Approaches to the coalition structure generation problem range from dynamic programming, to integer programming and stochastic search. However, they are either not optimal or the problem becomes intractable when the instance's size is large. Rahwan et al. proposed a branch-and-bound based solution that is anytime and can return the coalition structure within a bound from the optimal [48].

Working in a coalition may require some coordination, among the robots, such as synchronizing item loading, controlling the speed and the direction of the robot during the transportation. Such issues are beyond the task allocation problem. In what follows, we present major MRTA through coalition formation.

2.7.1 Coalition Formation with Spatial and Temporal Constraints

Ramchurn et al. studied task allocation to multiple agents through coalition formation when: team work is synergistic, tasks have deadlines and workloads, while the agents have work capacities and need to travel to tasks [23, 3]. The problem, Coalition Formation with Spatial and Temporal Constraints, falls within the MRTA category ST-MR-TA:TW/Ha, according to our taxonomy [21]. Nonetheless, the problem further assumes that multiple groups of robots may work on a same task, one after the other. The problem is motivated by emergency response application, where a prompt allocation is required and where the number tasks is very important compared to the number of agents, to an extent which does not enable to handle all the tasks. Consequently, the coalition formation goal is to maximize the number of tasks that can be completed by their deadlines. Ramchurn et al. demonstrated that the problem is NP-hard; they have experimentally shown that an exact solution they devised, a mixed integer program, though optimal is not practical to solve even problem instances with small sizes (more than two hours of computation were necessary to solve problem instances with no more than two agents and seven tasks). Thus, the authors heuristically approached the coalition structure generation problem [23].

Ramchurn et al.'s tackled the coalition formation problem with a heuristic that can return suboptimal but prompt solutions. The greedy heuristic does a one step look ahead search to improve the allocation decision at each iteration. Specifically, it calculates the smallest coalition size necessary to complete each pending task; based on which it computes the best coalition (i.e., the one with the highest coalition value) to complete the task. Finally, the task that can be completed the earliest is allocated, if it enables more task reachability in the next allocation step [23].

The heuristic is experimentally evaluated on a synthetic problem instances that are sampled with uniform distributions. The proposed heuristic, earliest completion first with one step lookahead, allocates tasks more than the baseline when the number of agents is small or moderate. Though comparable to the baseline, when further agents are available, Ramchurn et al.'s heuristic turns out to reduce the agents' total travel time [23]. Although the authors claim that their heuristic is anytime and performs fairly well, it is worth to mention that: (i) only a partial solution is obtained if we interrupt the search at some iteration before the heuristic finishes, since it allocates one task per iteration; (ii) the baseline is a basic heuristic, for it simply allocates the best coalition with the minimum necessary size to complete the task with the most urgent deadline.

2.7.2 Decentralized Coalition Formation with Spatial and Temporal Constraints in Dynamic Environments

The coalition formation with spatial and temporal constraints problem [23] (discussed above) is formulated as a decentralized constraint optimization problem and extended to dynamic environments, where the tasks and the agents may change during the allocation [3]. The authors adopt an approach to coalition formation which decisions are local, to: (i) overcome the single point of failure issue; (ii) reduce the communication overhead; (iii) enable scalability and adaptation to changes inherent in dynamic environments. Furthermore, such distributed local decisions reduce the problem complexity. In practice, such an

approach is inherently suitable for the author's motivating application, a RoboCup Rescue scenario where the computation and communication resources are limited and unreliable and where the tasks may change or new tasks may appear.

The authors' solution, a decentralized constraint optimization algorithm [3] could allocate more tasks than the decentralized game theory based approach of Chapman et al. [41], while being less efficient than the centralized coalition formation heuristic [23]. When there is disruption in the environment—such as the appearance or disappearance of tasks and the arrival or the departure of agents during task execution— the authors' solution shows robustness compared to the baseline: it cuts down the number of exchanged messages and their total average size, while reducing the computation cost.

2.7.3 Dynamic Coalition Formation with Spatial and Temporal Constraints and Decaying Rewards

Amador et al. studied multi-agent task allocation, possibly through coalition formation, for the Law Enforcement Problem (LEP) [4] (Section 2.3.3). Similarly to previous problems ([23, 3]), LEP assumes workload and time constraints on tasks. However, the deadlines are soft to allow tasks to finish late, but with a decayed utility. The problem is ST-MR-TA:TW/So according to Nunes et al. taxonomy [21]. The tasks have different priorities; a low priority task can be interrupted in case a more important task appears. If so, in dynamic environments, a penalty is inflicted. The coalitions values depend solely on the number of the agents that form the coalition.

The authors tackle the coalition formation problem with a heuristic that is at worst polynomial and pseudo-polynomial time [3] in centralized and distributed settings, respectively. In particular, Amador et al.'s solution features a Pareto optimality that maximizes the allocation utility, while it balances the workload among the agents. In fact, since the tasks are in practice assigned to police officers, the allocation approach is concerned with fairness by finding assignments that would not lead to some officers envying others [4].

2.7.4 Coalition Formation with Physical Interference and Soft Deadlines

Guerrero et al. studied coalition formation in time sensitive scenarios, where the physical interference among the robots affects the coalition value [49]. Similarly to Ramchurn et al.'s work [23, 3], they consider tasks that are spatially dispersed, have workloads and deadlines, while the robots, as well as the coalition they may form, have different work capacities. However, task completion is rewarded according to some decaying function which the upper bound (i.e., the maximum reward) is obtained along the time interval that does not exceed the task's deadline (i.e., the tasks have soft deadlines, as opposed to hard deadlines in [23, 3]). The coalition value, which is ideally additive; is estimated based on the robots individual work capacities and a deviation that results from physical interference.

The coalition formation problem goal is to maximize the total reward collected from performing the tasks. To do so, the robots have to be teamed in coalitions that can complete tasks ideally by their deadlines, or as soon as possible afterwards. Guerrero et al. solve

[3] a numeric algorithm runs in pseudo-polynomial time if its running time is a polynomial in the numeric value of the input, but not necessarily in its length

their coalition structure generation problem by estimating the task's execution time based on a model of interference. The interference, which is physical, results from many robots accessing the same point simultaneously. In the application scenario [49], interference occurs when many robots simultaneously unload the transported objects at the common delivery point. Specifically, the considered application is about foraging tasks: a group of robots, which are randomly dispersed, have to transport some objects, which are dispersed on some bounded area, to a common delivery point. A robot task consists in loading, transporting and unloading an object.

The tasks are allocated with double round auctions. The interference prediction is performed both online and off-line, based on the support vector regression model. It is used to estimate the coalition value (the group work capacity) and consequently the task execution duration. Knowing the task's deadline, the task's duration estimation enables to form a coalition with an adequate size such that to collect the best possible reward. The problem falls within the category ST-MR-TA:TW/So, according to Nunes et al. MRTA taxonomy [21].

2.8 Conclusion

This chapter concludes the background study of our thesis. We have defined and illustrated the MRTA problem. We have shown some major application examples. Then, we presented existing taxonomies that distinguishes between the allocation assumptions and requirements, particularly Nunes et al.'s to which we have contributed [21]. Afterwards, we have presented some problems that bears similarity with multi-robot task allocation, then we gave an overview of the approaches that deals with the different MRTA problems in the literature. In particular, we elaborated on MRTA problems and approaches which are based on coalition formation. Based on this background study, we can now: (i) state the multi-robot task allocation problem we are interested in and show its utility; (ii) study the problem complexity, its category and its relation to the MRTA literature; (iii) Finally, propose and justify an approach which we can adequately deal with our problem.

Chapter 3

Multi-robot Task Allocation with Spatial Temporal and Capacity Constraints Problem

> The formulation of the problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill
>
> Albert Einstein

This chapter states the multi-robot task allocation problem that interests us. Besides, it studies the computational complexity, relates the problem to the literature and justify anytime non-exact approaches. In Section 3.1, we give a real life application that motivated this research. In Section 3.2, we formally state our problem while we illustrate through an emergency response scenario in the aftermath of an earthquake (the motivating application). In Section 3.3, we study the problem's complexity to see how to appropriately approach it. Finally, in Section 3.4 and 3.5, respectively, we relate our problem to routing, scheduling and MRTA literatures. Section 3.7 concludes the chapter.

3.1 Motivating application

Before stating our MRTA problem, we detail the application that motivates us and through which we illustrate the problem. We are interested in the emergency response that takes place in the aftermath of a serious earthquake when it hits an urban area, say a city like Algiers (Algeria). In a typical scenario, buildings collapse and block the roads with debris which may, as well kill, or injure civilians and ignite fires. The fire-fighters extinguish fires as soon as possible to save lives, to facilitate the evacuation of trapped civilians, to limit material damages and to stop the fire from spreading to neighbour buildings. The ambulances evacuate injured civilians to hospitals, eventually promptly so to save their lives or limit health decay. The police clear the roads, from debris, to fluidize the traffic of fire-fighters, paramedics and civilians vehicles.

The coordination of the rescuers is necessary to respond effectively to the disaster. It

can intuitively be seen as task allocation to multiple physical agents. When the earthquake is severe, the number of rescue tasks is likely large relatively to the number of rescuers. That is, the numbers of: collapsed buildings, civilians injured, blocked road and fires should be beyond the capacity of the police, fire-fighters and paramedics. Since an emergency rescue task has a deadline or increases in intensity, tasks would fail if not handled on time. Thus, the rescuers should be allocated such that to handle a maximum of tasks, for instance, by prioritizing the urgent ones or by searching for the most effective rescue schedule. Rescue tasks may have a precedence or simultaneity constraint, such as to dig out a trapped civilian before providing first aid.

3.2 The Problem Statement

We state our multi-robot task allocation problem while we use emergency response as the running example. We assume a set of embodied agents that can travel to execute a set of tasks. The agents can be robots, terrestrial vehicles, aerial vehicles (drones) or a team of humans that need to be coordinated systematicallyto effectively execute the tasks. The tasks have some workload and time requirements, while they share the same importance. A task is worth to be handled if it can be completed by its required time (i.e., a hard deadline). The agents have a limited work capacity. The agents and the tasks are geographically dispersed. Tasks don't relocate, while agents do. We assume an environment static and observable. That is, the agents and the tasks are known upfront and do not change during the allocation time. We assume that an agent does at most one task a time, a task requires only one agent and tasks are assigned over extended time horizons. For simplicity, we assume one single type of task that requires a single type of agent (or one capability) and tasks which are independent.

According to Nunes et al. taxonomy [21] (cf., Section 2.4.3), the MRTA problem we have informally defined, above, falls within the "ST-SR-TA/TW:Ha deterministic" category. In what follows, we formally state and illustrate the problem.

3.2.1 Basic Definitions

Let $A = \{a_1, \ldots, a_n\}$ be the set of agents. Each agent $a \in A$ has an initial location $l_a^0 \in L_A$ at time $t = 0$, where L_A is the set of the agents' initial locations on the Euclidean plane. In our example, the agents maybe the firefighters, which are initially located at their station or somewhere in the city by the time fires unveil. Let $K = \{k_1, \ldots, k_m\}$ be the set of tasks. A task $k \in K$ has a fixed location $l_k \in L_K$, a deadline d_k beyond which k is lost if not handled, and a workload w_k that is an estimate of the amount of work necessary to complete handling k. For instance, a fire extinguishing task is located at a building on fire. The buildings' material and area, the weather condition and the fire duration (so far) can be used to estimate: how long the building would withstand the collapse (i.e., the deadline) and the amount of work to put out the fire (i.e., the workload).

Each agent a has a capacity p_a equals to the number of work units that a can perform in one time unit. For instance, a firefighters brigade capacity can be the fire's area it can extinguish in one minute. Difference in capacity stems from skill, experience or the possessed tools. A firefighters brigade maybe more efficient then another when it posses

a bigger water tank with many hoses or high pressure. We consider a discrete time, we assume that the agents' travel and task execution times are measurable (e.g., in seconds or minutes) and a reference time $t = 0$. We define $\rho : (L_A \cup L_K) \times L_K \to [0 \dots \infty]$, the agent travel time function. For simplicity, we assume that paths are free of blockages and traffic jams, the agents have the same constant velocity and ever take the shortest path between two consecutive tasks. So $\rho(.)$ is linearly proportional to the travel distance.

3.2.2 Problem Constraints

The tasks have to be allocated without violating any spatial, temporal or capacity constraint. The spatial constraint restricts where a task can be handled and incurs a travel cost to the agents. The temporal constraint restricts the time by which a task can be reached and completed, while the capacity constraint restricts the number and the duration of visits to tasks. An assignment of a task to an agent is said "feasible" if it respects the spatial, temporal and capacity constraints. Formally, the assignment of agent a to task k at time t is feasible, if and only if the following spatio-temporal and capacity constraint holds true:

$$t + \rho(l_a^t, l_k) + \lceil w_k / p_a \rceil \leq d_k \tag{3.1}$$

where l_a^t is the location of agent a at time t (the assignment's time reference). That is, at any moment an agent should only attempt a task it can reach and finish by its deadline, given the agent's current position, capacity and the task's workload. Furthermore, the agent's assignment periods cannot overlap, nor can a task be executed by more then a single agent. So the assignments should be mutually exclusive with regard to tasks, even for the same agent since we assume that task execution is no pre-emptive.

3.2.3 Search Space

The search space, S, is the set of all feasible assignments. Let $\tau_{t_1,t_2}^{a \to k}$ be agent a's assignment to task k during the period $[t_1, t_2]$ (we abuse the notation to mean: $\{t_1, t_1 + 1, ..., t_2\}$)

$$\begin{aligned} S = \{\tau_{t_1,t_2}^{a \to k} \mid \quad & t_1 = t + \rho(l_a^t, l_k),\ t_2 = t_1 + \lceil w_k / p_a \rceil, \\ & t_2 \leq d_k\}_{a \in A, k \in K, t \in \{0,\dots,d_k\}} \end{aligned} \tag{3.2}$$

That is, for each agent and task pair, (a, k), we generate all the feasible assignments over the time window $[0, d_k]$ (i.e., when task k maybe allocated). Since an agent a travels to task k, the arrival time t_1 depends on a's current location (at time t, which corresponds to zero in a's first assignment or the time by which a finishes the previous assignment). The time t_2 (by which a completes k) should meet k's deadline, for the assignment $\tau_{t_1,t_2}^{a \to k}$ to be feasible (Eq. 3.1). A candidate solution, denoted by Γ, is a subset of S such that:

$$\begin{aligned} \Gamma = \Big\{ \tau_{t',t''}^{a \to k} \in S \mid \forall \tau_{t_1,t_2}^{a_1 \to k_1}, \tau_{t_1',t_2'}^{a_2 \to k_2} \in \Gamma : k_1 \neq k_2, \\ a_1 = a_2 \Rightarrow [t_1, t_2] \cap [t_1', t_2'] = \emptyset \Big\}. \end{aligned} \tag{3.3}$$

If the first condition holds and $a_1 \neq a_2$, Eq. 3.3 implies that any task is at most assigned to one agent. It implies when $a_1 = a_2$ that an agent is assigned to only one task at any moment. The second condition, with an abuse of "$\cap$" notation, ensures that an agent's assignment periods do not overlap, for the spatial constraint and the single-task robot allocation model adopted. The search space $\mathbf{\Gamma} = \cup\Gamma$.

3.2.4 Objective Function

The goal of our multi-robot task allocation problem is to maximize the number of tasks which can be completed by their deadlines. This is expressed as follows:

$$\text{argmax}_{\Gamma \in \mathbf{\Gamma}} \sum_{k \in K} \delta(k, \Gamma), \tag{3.4}$$

where Γ is a candidate solution constructed as defined in Eq. 3.3 and $\delta(.)$ is the task success function for a given solution, defined as follows:

$$\delta(k, \Gamma) = \begin{cases} 1 & \text{if } \exists a \in A,\ t,\ t' : \tau_{t,t'}^{a \rightarrow k} \in \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The function $\delta(.)$ returns 1 only if task k can be completed in the schedule defined by the assignments in Γ. This function can be seen as the task's completion reward function, the reward is unitary and common, thus the tasks have the same importance.

To the best of our knowledge, the problem we have just defined has not been "specifically" addressed. In what follows, we refer to it as multi-robot task allocation problem with spatial, temporal and capacity constraints (MRTA-STC). It is worth to mention that while we use the term robot, we do not exclude other type of agents, such as humans rescuers in emergency response or a fleet of aerial vehicles in a surveillance mission.

3.3 Complexity Analysis

We show that our problem MRTA-STC falls within the class of NP-hard problems. In doing so, we justify an approximate algorithm or a heuristic solution. Let us consider the following problem. Given n nodes on the Euclidean plane, where each node $i \in \{1 \dots n\}$ is assigned a score $\sigma(i) \geq 0$ with $\sigma(1) = \sigma(n) = 0$, find a route of maximal score through the nodes that starts at node 1 and ends at n, and which duration is no longer than a given t_{max}. This problem, known as the Orienteering Problem (OP), has been shown to be NP-hard. Consequently, so is the special case of OP with unit scores [50]. Then, we can conclude that MRTA-STC is NP-hard, if we reduce it to the special case of OP.

Theorem 1. *MRTA-STC is NP-hard.*

Proof. Consider the following single-agent version of MRTA-STC. The tasks correspond to the subset of nodes $\{i | i \neq 1, i \neq n\}$, node 1 corresponds to the agent's initial location and node n corresponds to a given node on the Euclidean plane. The score of nodes $\{i | i \neq 1, i \neq n\}$ is of a unitary value ($\sigma(i) = 1, \forall i \neq 1, i \neq n$) and, therefore, it corresponds to the reward of task completion (*cf.,* Section 3.2.4). There is a null score on the initial and

terminal nodes ($\sigma(1) = \sigma(n) = 0$), because we do not assume tasks at these two locations. The travel time between any nodes i and j, noted here t_{ij}, is constant (*cf.*, Section 3.2.1). The workload and deadline settings are, respectively, $w_k = 0$ and $d_k = \max\{t_{max} - t_{kn}, 0\}$ for every task k, where t_{kn} is the travel duration from node k to n. That is, workloads are set such that the agent earns the score as soon as it reaches the task, like in OP; whereas, deadlines are set such that the agent can reach the terminal node from any "visited" task, by the time budget t_{max}. For instance, suppose that agent a can reach task x, from its current location, by time 10, $t_{xn} = 5$ and $t_{max} = 14$. If a goes to x, then a reaches n by 15 which is beyond t_{max}. Yet setting $d_x = 9$ (earlier then the time by which a can reaches x) prevents a from visiting x. It is clear that any feasible solution to the special case MRTA-STC is also feasible to the corresponding unitary-scores OP. Therefore, if we have a polynomial algorithm that solves the former, then we could find a route of maximum score that starts at node 1 and ends at node n by t_{max}, thus, solving the unitary-score OP in a polynomial time. However, since OP is NP-hard, MRTA-STC is at least as difficult; since even its simple version – with a single agent and zero workload tasks – is so. □

3.4 MRTA-STC is related to the TOPTW

We show here that our MRTA problem bears similarity with the Team Orienteering Problem with Time windows (TOPTW) [43], a generalization of the OP (Section 3.3). In TOPTW, a set of m locations is given: each location $i = 1 \dots m$ is assigned a score $\sigma(i)$, a service duration T_i and a time window $[o_i, c_i]$ during which the service can start. The starting point (location 1) and the ending point (location n) of every tour are fixed. The time t_{ij} to travel from location i to j is known for all locations. Like for OP, not all locations can be visited but only those that are part of the tours that end by time t_{max}. A time window may impose a waiting time on the visit, as they may make it unfeasible (i.e., when the arrival time to the node is after the window closing time c_i). The location score can be earned at most once. The goal of TOPTW is to find a set of n tours, one per agent, that maximizes the total score and satisfies the time budget and the visits' time window constraints. We show how MRTA-STC is similar to the TOPTW and how it differs.

MRTA-STC has the same objective function as the unitary-score TOPTW and the same node score function (i.e., a node score is of a unitary value and earned at most once). Contrary to the TOPTW's time windows, which constrain the visit's starting time, MRTA-STC's time windows (of the form $[0, d_i]$) constrain the time by which a visit i must be finished (i.e., a deadline). In MRTA-STC the initial node differs among agents and the final node may be any task node. There is no time budget in MRTA-STC, yet any tour could be no longer than $d_{max} = \max_{k \in K} d_k$, which is the deadline of the task which completion can be the latest possible (i.e., $t_{max} = d_{max}$). Finally, task i's service time depends on the capacity of the serving agent a ($T_i = \lceil w_i / p_a \rceil$), unlike TOPTW where T_i is constant. Therefore, we should get new test instances for MRTA-STC problem. Nonetheless, we can generate these based on established TOPTW benchmarks.

3.5 How MRTA-STC relates to the MRTA literature

Perhaps the MRTA problem that bears most similarity with ours is Ramchurn et al.'s problem CFSTP [23]. Both MRTA-STC and CFSTP deal with MRTA when the tasks have spacial, temporal and workload constraints, while the robots have capacity constraints. Furthermore, both assume hard deadlines on the tasks, a number of tasks that is large compared to the number of the agents available and a deadline and workload setting that preclude the allocation of all tasks. Thus the aim to complete as many tasks as possible.

However MRTA-STC assumes single-robot tasks (i.e., agents do not work simultaneously on a single task). Nonetheless, assuming that the agents, instead of the coalitions, have different capacities implies that the agents have different utilities for a same task. That reduces the problem's complexity without reducing its utility. Contrary to CFSTP [23], we do not allow a task to be processed many times (i.e., by different agents at separate intervals), since we think that such a pre-emption is useless and adds an unwanted difficulty. We do not assume heterogeneous nor decaying rewards as in [4, 49], nor the possibility of splitting a task as in [49]. We do not assume interdependencies among the tasks or resource requirements for the agents as in [24]. Nonetheless, by considering the two latter assumptions, our problem becomes more realistic and utile for the motivating application—which should be the concern of a future research.

3.6 MRTA-STC Applications

MRTA-STC can mainly be applied to large scale emergency response coordination, such as extinguishing fires and rescuing civilians, in the aftermath of a serious earthquake at a city level. Our formulation can capture the problem of good or persons pickup and delivery with hard time constraints (such as mails service or radio taxis), if we substitute the work capacity with space or weight capacity on the vehicles. Besides, our problem is relevant to other domains such as resource allocation in Internet of Things, since it is a large scale system with tasks that are geo-localized, may require resources (e.g., computation power) not available locally and may change location—we may easily extend our multi-agent task allocation model to account for the last assumption.

3.7 Conclusion

In this chapter, we formally stated a utile MRTA problem (MRTA-STC), which we illustrated through an application scenario and we gave some other practical applications thereof. Moreover, we studied the problem's complexity and its relation to MRTA and the routing and scheduling literatures. We conclude that MRTA-STC is a novel problem, it is NP-hard CO and it bears similarity with the TOPTW and CFSTP problems. Consequently, we suggest to tackle MRTA-STC with some metaheuristic, similarly to most approaches dealing effectively with the TOPTW problem. Furthermore, though not optimal, metaheuristics provide an anytime response promptly and can improve the solution if given more time, which is relevant to resource scarce and time-critical applications such as the ours a. We also suggest to base our experimental comparison on TOPTW approaches

and benchmarks, particularly since our application domain may require to trade-off solution quality to promptly allocate the tasks. The following chapters present our attempts to effectively tackle MRTA-STC problem and the experimental results.

Chapter 4

Combinatorial Auctions with Stochastic Search Approach to MRTA-STC

In this chapter, we present a first attempt to address the MRTA-STC problem (Chapter 3). We formulate our task allocation problem as a combinatorial auction problem, where the bidders are the agents and the bids are sequences of tasks that the agents commit to do. In Section 4.1, we introduce combinatorial auctions as an artifact to sell goods in bundles in a market, but also for other purposes such as resource and task allocation. In Section 4.2, we briefly review the literature of combinatorial auctions, with an emphasis on their application to task allocation to multiple robots. Then we present our approach to the problem, MRTA-STC, using combinatorial auctions. We first show, in Section 4.3, how to generate bids for tasks. Then, in Section 4.3.3, we show how we clear the auction using a fast stochastic search, inspired by the work of Boughaci et al. [51]. Finally, in Section 4.4, we present some experimental results, a discussion and conclusion on the utility of approaching MRTA-STC with combinatorial auctions where the auction clearing is made with a metaheuristic.

4.1 Combinatorial Auctions

Combinatorial Auction (CA) is a market mechanism to sell goods when the buyers value items grouped altogether. Contrary to single item auctions, in CA the buyers place bids on bundles of discrete items (or packages). This sort of auctions is relevant when some of the items (for sell) present complementarity or substitutability, for the buyers. An example of complementarity is when an airline company requires two adjacent gates, at a specific time, at some airport; therefore, it is not willing to give any value for obtaining one gate only. An example of substitutability, is when a buyer bids on a single copy of a book for some price, while he/she bids for many copies with a lower unit price. Furthermore, CA maybe more convenient for the sellers, since selling items in bundles is prompter than running multiple single item auctions. This is the case in estate sales, for instance, where a substantial portion of the material that belongs to a deceased (or moving) person must be disposed quickly.

The combinatorial auction problem can be stated as follows. Let $G = \{g_1, \ldots, g_m\}$ be the set of goods (items) to sell. Assume we have a group of n buyers that can issue n bids on the items, one bid per buyer. The set of bids is then $B = \{b_1, \ldots, b_n\}$, where b_i is the bid issued by buyer i. A bid b_i is composed of: (i) a bundle that gather some of the goods for sell (i.e., $b_i \subseteq G$) and (ii) an offer o_i that is the price buyer i is willing to pay for the bundle ($o_i \geq 0$). The goal is to find a set of bids, $W \subseteq B$, which bids are mutually exclusive (i.e., they do not share any item), such that to maximize the seller's revenue. Formally speaking, let $A_{n \times m}$ be a two dimensions matrix that represents the bids. An element $a_{ij} = 1$ iff item j is part of bid i; otherwise, $a_{ij} = 0$. Let X_n be the auction outcome matrix. That is, $x_i = 1$ iff bid i wins the auction, $x_i = 0$ otherwise. Clearing the auction is referred to as Winner Determination Problem (WDP). It can be stated using the following integer program [51]:

$$\begin{aligned} & \text{Maximize } \textstyle\sum_{i=1}^{n} o_i x_i \\ \textbf{Such that}: \quad & \textstyle\sum_{i=1}^{n} a_{ij} x_j \leq 1 \qquad j \in \{1 \ldots m\},\ x_i \in \{0, 1\} \end{aligned} \tag{4.1}$$

Because bundles often overlap, clearing the auction (i.e., solving the WDP) is an optimization problem that requires searching many possibilities. The WDP is intractable when the problem instances have a large size, since it has been proved that the problem is NP-complete [52]. A number of complete algorithms—which guarantee optimality of solution—have been proposed for WDP, including: dynamic programming [53], algorithms that consider problems with special structures [1] and AI-style search techniques, such as deep first search [54]. The latter uses clever structuring of the search space, preprocessing, heuristic ordering methods and pruning techniques (Branch-and-Bound), that allow the search to find optimal allocations rather effectively. Furthermore, Fujishima et al. algorithm for CA [54] exhibits reasonable anytime performance: it provides good allocations prior to finding the optimal ones.

When the problem instances are large/difficult, or when the solutions are needed quickly, existing algorithms for CA are likely to prove inadequate. In many, if not most, resource-allocation or E-commerce problems that are most readily modeled as CAs, it seems apparent that real-time response to very large problems will be expected [52]. Complete algorithms necessarily spend considerable time "proving" that the solution they produce is optimal, at the expense of providing high-quality (though perhaps suboptimal) solutions quickly. Furthermore, as problem instances become larger, complete algorithms will, in most cases, become infeasible. While certain domains may require optimal solutions (e.g., for legal reasons), it is expected that typical applications of CAs will be ideally suited for techniques that produce high-quality, approximate solutions quickly (or within a suitable time frame) [52].

Drawing an analogy to scheduling research, for example, large scheduling problems – such as ours – are invariably solved heuristically. Incomplete solutions to CA include: metaheuristics [52, 55], which resemble multi-restart local search, hybrid metaheuristics,

[1] Such as when bids have a limitation on size or a restriction on the structure, since doing so allows one to obtain polynomial time algorithms.

such as [56] which combines Branch-and-Bound search with simulated annealing. While such methods do not guaranty the optimality of the solution, they empirically proved to be able to promptly find high quality, sometimes optimal, solutions on difficult WDP instances [52, 51].

4.2 MRTA-STC through Combinatorial Auctions

Approaching MRTA problems through CA has been considered in early works on urban search-and-rescue [33, 57]. Nair et al. addressed the task allocation problem in RoboCup Rescue [2], where tasks arrive dynamically and can change in intensity during the allocation [33]. In the considered scenario—disaster mitigation in the aftermath of an earthquake in a city—, the fire station, the ambulance center and the police office take on the role of the auctioneer, whereas the fire brigades, the ambulances and the police forces take on the role of bidders. The authors clear the auction with a centralized depth-first branch-and-bound search [9] that branches on bidders. Nonetheless, the search is sub-optimal, since it restricts the number of bids that are generated. Suarez et al. uses the MAGNET framework for CA [58] to allocate tasks in RoboCup Rescue Simulator. To deal with the dynamic environment, the authors implemented a task re-scheduling algorithm [57], which re-allocates a task if it fails as when the path get blocked.

Knowing that MRTA-STC problem is NP-Hard (cf., Section 3.3), we may need to trade off solution optimality with solution cost (i.e. to get prompt response times). Similarly to [33, 57], we approach MRTA-STC problem with CA. We propose and study an incomplete algorithm that combines problem-dependent search pruning (as [33, 57]) and stochastic search moves, to rapidly explore the solution space [59]. To do that, we address both the Bid Generation Problem (BGP) and the WDP. The former is concerned with generating feasible bids on tasks for each agent, as we explain in the following section.

In the aftermath of a disaster, a set of rescuers (field agents) rover in the disaster area to search for rescue tasks. The rescue coordinator (central agent) is informed about the positions of the rescuers and the discovered tasks: their location, workload estimation and deadline estimation. Then, this latter issues a call for proposal to the rescuers, to split the pending tasks among them. An agent bid contain an ordered sequence of tasks, which he can do, and an offer that includes the cost and the value of performing that sequence of tasks.

Formally, we define an agent a's bid, noted b_a, as follows:

$$b_a = [\langle k_{a1}, k_{a2}, \ldots, k_{ah_a} \rangle \,;\; o_a] \tag{4.2}$$

Where k_{ai} is the i^{th} task that agent a commits to do if its bid wins, o_a is the bid's offer (Eq. 4.3) and h_a is the length of the bid (i.e., the number of tasks to do, $h_a \leq m$). In accordance with our objective function, we define the bid value as the number of tasks the agent commit to do (Eq. 4.4), while we define the bid cost as the total travel distance that

would be necessary to visit the bid's tasks (Eq. 4.5),

$$o_a = (value_a,\, cost_a) \tag{4.3}$$

$$value_a = |\,\langle k_{a1}, k_{a2}, \ldots, k_{ah_a}\rangle\,| \tag{4.4}$$

$$cost_a = \sum_{j=0}^{h_a-1} dist_j^{j+1}, \tag{4.5}$$

where $dist_j^{j+1}$ stands for the travel distance—or the travel time, since we assume a velocity that is constant and common among the agents—between two sequential tasks, k_{aj} and $k_{a(j+1)}$, of bid b_a. We put $dist_0^1$ as the distance between l_a^0 (the initial location of agent a) and k_{a1} (the location of the first task that agent a offers to do). We define the best bid of agent a, b_a^*, as the one with the highest bid value among the other possible bids. Specifically, it is the one with the longest sequence of tasks (Eq. 4.6), B_a are agent's a possible bids.

$$b_a^* = \text{argmax}_{b \in B_a}\ length(b) \tag{4.6}$$

As pointed out earlier, it is likely that the bids submitted by the many agents, are in conflict (not mutually exclusive with regard to tasks). Even more, since supposedly each agent tries to maximize the number of tasks it can take. Therefore, submitting the best bids is not practical, for when each agent submits its best bid, clearing the auction, even optimally, doesn't necessarily yield the globally best allocation. Instead, an agent should submit all its feasible bids. However, doing so causes an important communication overhead, which is not affordable or not desirable in a disaster scenario. Alternatively, we delegate to the central agent the problem of bid generation. To avoid combinatorial explosion, we bound the bid length by some empirical value h_{max}, such that $\forall a \in A : h_a \leq h_{max}$. Furthermore, we set a distance horizon, $distH$, and a time horizon $timeH$ to dismiss tasks far away and tasks with very late deadlines during the BGP.

To find a task allocation, we clear the auction by selecting a set of bids that cover the maximum number of tasks (maximum total value), while having the smallest total travel distance possible (lowest total cost) when there are many bids with equivalent values. The task allocation is operated by the central agent as follows:

1. gather informations about pending rescue tasks;
2. generate feasible bids for each available field agent from the set of pending tasks.
3. find the winning bids and allocate the agents to the tasks;
4. Re-iterate (go to (1)), whenever there are tasks left—either not considered tasks (in the last iteration) or tasks that showed up— and agents available.

4.3 Bid Generation Problem

Given n bidders (agents) and m items (tasks), the BGP search space is $O(nm^m)$. Difficult MRTA-STC problem instances have a large number of tasks, compared to the number of

agents. Consequently, the number of bids grows exponentially in the number of tasks. It is then not practical to exhaustively generate the bids. To reduce their number, we bound the bid length with a limit h_{max}. Then, in the worst-case we would have a $O(nm^{h_{max}})$ BGP. In practice, the search space is further down sized by the problem constraints. The limit h_{max} is a free parameter in the BGP algorithm which setting is left to the user. For instance, where tasks may appear or change during the allocation, small values for h_{max} (say $h_{max} \leq 10$) are suitable, because tasks that are scheduled for late execution are likely to be re-allocated and BGP would be solved prompter. By centralizing the BGP solving, we reduce the communication overhead – in the worst-case – from $O(nm^m)$ to $O(nm)$, since at each allocation round, an agent exchanges two messages only (the list of discovered tasks and the sequence of allocated tasks) with the central agent.

4.3.1 Generate Feasible Allocations

Algorithm 1: Find an agent's task assignments that are feasible at some moment

Input: $a, t, K^t_{pending}, timeH, distH$
Output: F^a_t

1 $F^a_t \leftarrow \emptyset$
2 **forall** $k \in K^t_{pending}$ **s.t** $distance(l^t_a, l_k) \leq distH$ **and** $d_k - t \leq timeH$ **do**
3 $\quad t_s \leftarrow t + \rho(l^t_a, l_k)$
4 $\quad t_e \leftarrow t_s + \lceil w_k / p_a \rceil$
5 $\quad$ **if** $t_e \leq d_k$ **then** $F^a_t \leftarrow F^a_t \cup \{\tau^{a \to k}_{t_s, t_e}\}$

An agent's bid is composed of a sequence of feasible task assignments. To generate a single bid for an agent a, we repeatedly look for the task allocations that are feasible at many times (t). Let F^a_t be the set of alternative feasible allocations for agent a at point of time t. The set $K^t_{pending}$ contains tasks that are not yet allocated at time t. Again, $\tau^{a \to k}_{t,t'}$ denotes that agent a works on task k during the time frame $[t, t']$. We generate the set F^a_t as follow (Algorithm 1). When we run through the set of unallocated tasks (line 2) while not considering far away tasks (beyond $distH$), nor tasks with late deadlines (beyond $t + timeH$). The free parameters, $distH$ and $timeH$, reduce the number of the alternative allocations at any moment; consequently, they help later, along with h_{max}, to further reduce the number of bids to generate. A task assignment's start time (line 3) is calculated based on the agent's location at the current time (t). The assignment's finish time (line 4) is calculated based on the assignment's start time, the task's workload and the agent's capacity—since time is discrete, we do an integer division to compute the allocation period ($\lceil w_k / p_a \rceil$). The assignment is generated if, and only if, it can be fulfilled by the task's deadline (line 6).

4.3.2 Generate Bids

We generate the set of bids for every agent (Algorithm 2, which we refer to as BGP pruning or BGP-P). Our approach to the BGP prunes the search. BGP-P recursively builds the set of bids, B_a, for every given agent a based on some bid root r, (initially empty). Algorithm 2 runs recursively until: the bid maximum length is reached (line 2), no task is left unallocated or no further task assignment is feasible for the agent (line 5). At each call, Algorithm 2

Algorithm 2: Generate an agent's feasible bids (BGP-P)

Input: $a, t, K^t_{pending}, r, timeH, distH, h_{max}$
Output: B_a, agent a's set of feasible bids

1 **if** $r = \emptyset$ **then** $B_a \leftarrow \emptyset$
2 **if** $length(r) + 1 = h_{max}$ **then return**
3 $F^a_t \leftarrow$ alternative feasible agent a's assignments (Algorithm 1 with input: $a, t, K^t_{pending}, timeH, distH$)
5 **if** $F^a_t = \emptyset$ **or** $K^t_{pending} = \emptyset$ **then return**
6 **forall** $\tau^{a \to k}_{t',t''} \in F^a_t$ **do**
7 $b_a \leftarrow r \cup \{\tau^{a \to k}_{t',t''}\}$ `/* a new bid that stems from the current bid root */`
8 Compute the value and cost of bid b_a (Eq. 4.5, 4.4)
9 $B_a \leftarrow B_a \cup \{b_a\}$
10 $K^{t''}_{pending} \leftarrow K^t_{pending} \setminus \{k\}$
11 Call Algorithm 2 with $(t, r) = (t'', b_a)$ `/* The bid` b_a `becomes the bid root */`

finds the alternative feasible assignments for the agent (a) at the current allocation reference time t (line 3). Based on these, we generate a number of new bids (b_a) that have bid r as a root (line 6,7). Each time, we compute the new bid cost and value, we add the bid to the agent bids list, B_a and we update the set of pending tasks (lines 8,9 and 10). Effectively, each newly generated bid becomes a bid root for many coming bids and the added assignment's end time becomes the current allocation reference time in the next call of Algorithm 2 (line 11).

4.3.3 Clear the auction and get the task allocation

To approach our WDP, thus to find some solution to MRTA-STC, we replicate Boughaci et al.'s stochastic search for WDP [51]. Indeed, contrary to the authors' claim, the metaheuristic does not do a local search, as we show later. Interestingly though, the results that the authors report [51] show overall a performance, with regard to solution quality, that is better than the competitors' [52]. Furthermore, although the metaheuristic does not find the best-known solutions on the WDP benchmarks, it is worth to consider that it is prompt and simple to implement. We detail the implementation of the metaheuristic [51] for our problem, WDP-SS (Algorithm 3). We refer to it as WDP-SS (Winner Determination Problem with Stochastic Search).

The stochastic search starts from an arbitrary solution (Algorithm 3). Namely, for each agent, we generate the feasible bids (line 3), we randomly rank them (line 4), then we select the highest ranked bid (line 5) to include in the set of winning bids (line 6). The initial solution is cleaned from conflicts arbitrarily: given any two bids in conflict, we keep the one with the highest rank, until there is not conflict (line 7). Two bids are in conflict if they share the same agent or (at least) a same task—given that we adopt an ST-SR task allocation model. Then, we iterate many times to improve the current solution; either through finding and inserting the currently best bid (line 12) or by doing a random walk [60] (line 11), to escape a possible local area, and thus, to explore other neighborhoods which may contain a better solution. Either way, we update the set of winning bids and the set of remaining bids (lines 16,17, respectively). To decide on how frequently we should explore,

Algorithm 3: Clear the combinatorial auction with a Stochastic Search (WDP-SS)

Input: $K, A, timeH, distH, h_{max}, wp, maxIter$
Output: W, the set of winner bids.

1 $W \leftarrow \emptyset$
2 **forall** $a \in A$ **do**
3 $B_a \leftarrow$ agent a's feasible bids (Algorithm 2, with $K^t_{pending} = K, t = 0, r = \emptyset$)
4 $RK_a \leftarrow$ a mapping of *random keys* to bids $\in B_a$
5 $b^*_a \leftarrow \text{argmax}_{b \in B_a}\ RK_{a,b}$ `/* an arbitrary bid among agent a's bids */`
6 $W \leftarrow W \cup \{b^*_a\}$
7 Clear conflicts in W such that to keep the bids with the highest RK values
8 $B \leftarrow \bigcup_{a \in A} B_a$ `/* define WDP search space */`
9 **for** $i \leftarrow 1$ **to** $maxIter$ **do**
10 $r \leftarrow$ a random real number $\in [0, 1]$
11 **if** $r < wp$ **then**
12 $b_s \leftarrow$ random bid $\in B$ `/* random walk move */`
13 **else**
14 $b_s \leftarrow \text{argmax}_{b \in W}$ {number of tasks in b} `/* best improvement move */`
15 **if** $\exists b \in W$ **s.t** $agent(b_s) = agent(b)$ **then**
16 $W \leftarrow W \setminus \{b\}$ `/* remove a bid previously selected for` b_s`'s agent */`
17 $B \leftarrow B \cup \{b\}$
18 $W \leftarrow W \cup \{b_s\}$
19 $B \leftarrow B \setminus \{b_s\}$
20 **forall** $b \in W$ **s.t** $conflict(b, b_s)$ **do**
21 $W \leftarrow W \setminus \{b\}$ `// clear bids in conflicts with the current selection`
22 $B \leftarrow B \cup \{b\}$

we use a random walk probability, wp, which is a free parameter that we empirically set, such that to have an appropriate frequency of random walks. When we select a new bid, we remove any previously selected bid by the same agent (line 13–15). Then, we clear (from the current solution) all the bids that are in conflict with the selected bid (lines 18–20).

The metaheuristic (Algorithm 3) is supposed to be a SLS where the LS is of the type "best-improvement first" [51]. However, as we noticed (later) during the empirical tests [59]: selecting the currently (not yet inserted) best bid, then clearing the bids conflicting with the selection (lines 12, 13-15, Algorithm 3), often do not result in a solution quality improvement—assuredly, the bids conflicting with the best selection are often not the least best bids in the current solution, and the best bid is likely to be in conflict with many. Nonetheless, as such, the computing overhead of the not random selection step – including the conflict clearing step – is reduced from $O(n^3 m^{h_{max}})$ (case of best-improvement LS [51]) to $O(nm^{h_{max}})$(case of best bid insertion, Algorithm 3).

4.4 Experimental Evaluation

We experimentally evaluate our market-based approach to MRTA-STC. We implemented the algorithms, we generated the test data and we set the algorithms free parameters. We did many experiments, each replicated several times to obtain an average performance. The problem instances generated differ in: (i) the number and positions of agents and tasks and (ii) the demands of the tasks (workload and deadline). The free parameters to set

are: the random walk probability and the number of iterations for the WDP-SS; the time and distance horizons and the bid maximum length for BGP-P. We opt for an average case study, by making no particular assumption on the problem structure. To do so, we choose uniform distribution to sample the agents' and tasks' locations, workloads and deadlines

4.4.1 Experimental Setup

We generate many problem instances with different structures. We consider an $x \times x$ (square) grid maps, on which tasks and agents are scattered uniformly at random, specifically $l_a \sim (U(0, x-1), U(0, x-1))$ and $l_k \sim (U(0, x-1), U(0, x-1)))$. We set the travel velocity at a constant and unitary value for all the agents ($velocity = 1$). Similarly, we set the agent (work) capacity at a unitary value for all agents ($p_a = 1, \forall a \in A$). To generate tasks workloads and deadlines values, we use uniform distributions over large intervals (to simulate wide ranging scenarios), respectively, as $d_k \sim U(5, 600)$ and $w_k \sim U(10, 50)$. The test instances are organized in problem structures coded as "UNI-n-m-x", where "UNI" refers to sampling with uniform distributions, n is the number of agents, m is the number of tasks and x is the map dimension. Each problem structure regroups 100 instances, labeled "inyy" (where yy is the instance number), obtained by sampling, as we explained above. The test instances are grouped in two sets: testset1 and testset2, which are respectively available online here: http://tinyurl.com/tastp12 and http://tinyurl.com/tastp13. We run the experiments on a Dell Inspiron laptop with 1.6 GHz Intel Centrino CPU, 1 GHz RAM and 2 MBytes cache memory. The machine runs under Linux Ubuntu 10.04 operating system. We coded the algorithms in Java and run them under Java Virtual Machine.

We tune our algorithms on the instances to test with. We set the free parameters based on trial-and-error and some common sense. Specifically, we set $h_{max} \approx \lceil m/n \rceil$, to balance the number of allocated tasks among the agents while reducing the BGP search space. Indeed, we obtained the best solution qualities when h_{max} is set likewise. We found the best setting for the time horizon as: $timeH = d_{max}$. We tested WDP-SS with many $wp \in (0, 0.33]$, since intuitively the random walk should be occasional at most. The best setting we found is $wp = 0.25$. We set $nrbIter = 3000$, to not exceed a real time response (less than one minute).

4.4.2 Results and Discussion

Problem	*distH* set at the *maximum* value			*distH* set at the *mean* value		
	Time	*Quality*	*Travel*	*Time*	*Quality*	*Travel*
UNI-3-5-20	**11.92**	4.41	**73.3**	12.48	**4.51**	74.69
UNI-3-8-50	794.94	4.81	**286.59**	**771.4**	**4.88**	301.49
UNI-3-12-100	**1436.42**	4.71	**511.21**	1480.93	**4.88**	534.6
UNI-4-9-50	7170.1	**4.92**	274.34	**6687.43**	4.66	**249.38**
UNI-5-12-100	13445.62	**4.83**	419.74	**24471.39**	4.78	**414.89**

Table 4.1: The average performance of BGP-P and WDP-SS (combined) with different configurations of distance horizons on MRTA-STC instances of *testset1*

In a preliminary evaluation, we only tested our CA algorithms on five arbitrary

problem instances pertaining to five different problem structures (testset1 [2]). Table 4.1 summarizes our results [59]. The first column from the left lists the problem structures, the following columns report the average performances measured over 100 run replications. Respectively, they report the computation time (in milliseconds), the solution quality (the number of allocated tasks) and the totally traveled distance. The first set of performance results (columns two to four, Table 4.1) is obtained by running WDP-SS when the distance horizon is set at the maximum value ($distH = 2x$) —i.e., an agent can see and bid on all known tasks in the disaster map. The second set of performance results (columns five to seven, Table 4.1) is obtained by running WDP-SS when the distance horizon is set at a mean value, which is the average distance between the tasks locations ($distH = \sum_{i=0}^{m-1}(\sum_{j=i+1}^{m} dist(k_i, k_j))/(m-i)$). Table 4.1 clearly shows that BGP-P combined with WDP-SS can not significantly improve solution quality by tweaking the distance horizon. The number of allocated tasks is always around 5, which is the best-found setting for h_{max}. Furthermore, we could not improve solution quality even beyond $10s$ of runtime (Table 4.1). When checking many solutions, we noticed almost a single winner bid. We think that is likely linked to the absence of a local search.

Problem inst.	$distH$ set at the maximum value					$distH$ set at the mean value				
	$\#bids$	$Time$	$Qual.$	$Qual_{max}$	$Travel$	$\#bids$	$Time$	$Qual.$	$Qual_{max}$	$Travel$
UNI-3-9-50	**2408**	294	7.44	**9.0**	294	238	**16**	**9.0**	**9.0**	**227**
UNI-3-12-50	**4316**	1190	**5.05**	7.0	**236**	2577	**524**	4.12	**8.0**	255
UNI-4-12-50	**3677**	860	3.73	7.0	**174**	535	**67**	**3.81**	**7.0**	190
UNI-4-16-50	**12263**	12524	4.12	8.0	**237**	4807	**1475**	**4.60**	**9.0**	250
UNI-5-15-50	**21209**	56675	4.6	6.0	228	4653	**1419**	**5.5**	**8.0**	**219**

Table 4.2: The average performance of BGP-P and WDP-SS, when configured with different distance horizons, on MRTA-STC instances of *testset2*

To further evaluate our CA approach, we further test on testset2 and we tune WDP-SS over all wp values. We want to check an hypothesis: WDP-SS doesn't do a LS and is thus similar to a mere random search. Testset2 problem instances are similarly generated totestset1's, except that grid map is 50×50, we vary $m \in \{3 \times n, 4 \times n\}$ for $n \in \{3, 4\}$ and we set $m = 15$ for $n = 5$. We repeat the two experiments done on testset1. We keep the time horizon, $timeH$, set at the maximum ($timeH = d_{max}$) and the other BGP-P parameters set as: $h_{max} = 4, velocity = p_a = 1$. For WDP-SS, we set $maxIter = 10^5$ and we tune wp on each problem instance. The results are reported in Table 4.2. Respectively from left to the right, Table 4.2 contains the problem structure, the number of bids considered, the average computational time, the average solution quality, the best solution quality (obtained after run replication) and the average total travel distance. We can see (Table 4.2) that WDP-SS and BGP-P with a mean distance horizon, compared when BGP-P is set with maximum distance horizon, fairly improve solution quality and solution computation cost. Furthermore, more tasks could be allocated if we replicate WDP-SS several times. Nonetheless, WDP-SS is either way like a random search (Figure 4.1). Effectively, we tuned WDP-SS over the whole range of wp (i.e., $[0.0, 1.0]$), using 0.05 increments. We noticed that generally WDP-SS is a little sensitive to the random walk probability (the solution

[2]available online at http://tinyurl.com/tastp12-test

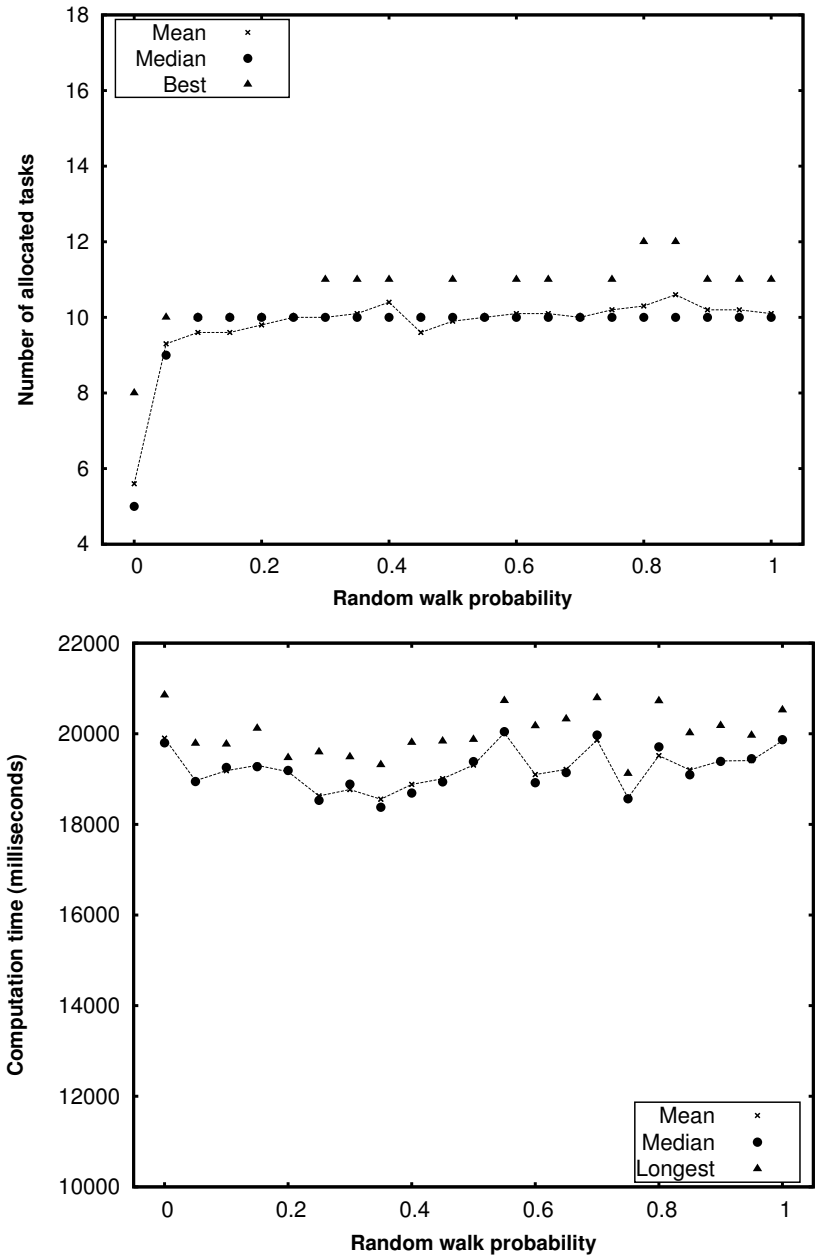


Figure 4.1: A WDP-SS tuning plot sample. (Top) solution quality evolution. (Bottom) computational cost *vs.* random walk probability. (instance UNI-3-18-50/in00, parameters setting ($maxIter = 5000$) and ($bidMaxLen = 9, distH = \text{mean}, timeH = d_{max}$)

quality tends to be the same). Even worse, sometimes WDP-SS tends to improve the solution quality as it shifts to a random search –i.e., as *wp* increases – (see the median and best plots, Figure 4.1) . Moreover, WDP-SS computational time is a little sensitive to *wp* change. Its tendency—a slight increases and decreases as *wp* increases—suggests that WDP-SS is as a random search (Figure 4.1). We did not do further experiments with a large number of tasks, since even with a few number of tasks, the results seem to have a low quality and a high cost.

4.5 Conclusion

This brief chapter summarizes our first attempt to approach MRTA-STC, using Combinatorial Auctions and metaheuristics. Specifically, we addressed the bid generation problem, using parameter-based search pruning, while we addressed the winner determination problem with a restart search that is adjusted by a random walk probability, to control the search tendency between exploitation and exploration. We obtained results that are unsatisfactory, nonetheless, the preliminary work we conducted was useful to get familiar with the problem and see in practice the important computational overhead (even on problem instances with an average size and using a incomplete search). Not the least, we discovered a shortcoming in published approach when we implemented it on our problem. We conclude that combinatorial auctions are not convenient for MRTA-STC (at least as we designed them), not so with regard to solution quality nor response time— not surprising, if we see the little works based on CA for MRTA. Consequently, the following chapters will investigate approaching MRTA-STC with other techniques (heuristics and metaheuristics).

Chapter 5

Basic Heuristics for MRTA-STC

> Pour faire vite, il faut aller doucement.
>
> Maurice Audin,

We describe, analyze and experimentally evaluate some basic heuristics to tackle MRTA-STC problem's complexity when the computation resource is limited or a prompt response is needed. Specifically, we study and build upon some routing and scheduling heuristics that are prompt, easy-to-implement and tuning free. We improve an existing heuristic and we propose a new one. We analyze theoretically the computational complexity of the presented heuristics, before empirically comparing them. In so doing, we offer solutions to MRTA-STC for devices with a limited computation power. Furthermore, we lay the ground for metaheuristics or hyper-heuristics, by offering the heuristics to embed within.

Section 5.1, 5.2 and 5.3 present three greedy heuristics: "earliest deadline first", "nearest task first", and "earliest completion first with one step look-ahead search". In Section 5.6, we present an optimized implementation of a classical insertion-based heuristic, the parallel insertion heuristic, to quickly construct a solution. In Section 5.7, we propose another parallel insertion heuristic, to improve solution quality, travel cost and response time. In Section 5.8, we empirically compare the proposed heuristics, on a synthetic testset with respect to response time, solution quality and travel cost.

5.1 Earliest Deadline First Heuristic

The Earliest Deadline First (EDF) heuristic, which is greedy, prioritizes urgent tasks. Specifically, it secures the allocation of tasks with urgent deadlines, while it "hopes" that enough time and agents would be available to allocate the remaining tasks. The heuristic is based on a scheduling rule of thumb that suggests to prioritize the most constrained.

We implement EDF heuristic as follows (Algorithm 4). We maintain in a vector, t^{free}, the times by which the agents become available. We iterates until no pending task can be allocated. At each iteration, we select the most urgent task, k^*, then we search for the closest agent a^* that is capable of completing the task (line 5–6). That is, the heuristic is time greedy when selecting the task, whereas, it is distance greedy when selecting the agent. We discard the current task if we cannot find an agent capable of finishing the task by its

deadline (line 7–8). Otherwise, we compute the task's assignment start and completion times, t^s and t^c respectively (line 10–12), add the assignment to the solution (line 13) and update the set of pending tasks and the time by which the assigned agent becomes available again (line 14,15).

Algorithm 4: Earliest Deadline First (EDF) heuristic

Input: K, A
Output: Γ^*

1 $\Gamma^* \leftarrow \emptyset$
2 $K_{pending} \leftarrow K$ `/* pending tasks */`
3 $t^{free} \leftarrow \langle 0, \ldots, 0 \rangle$ `/* agents availability times */`
4 **while** $K_{pending} \neq \emptyset$ **do**
5 $\quad k^* \leftarrow \text{argmin}_{k \in K_{pending}}\ d_k$ `/* earliest deadline task */`
6 $\quad a^* \leftarrow \text{argmin}_{a \in A}\ \{\rho(l_a^t, l_{k^*}) \mid t = t_a^{free},\ t + \rho(l_a^t, l_{k^*}) + \lceil w_{k^*}/p_a \rceil \leq d_{k^*}\}$ `/* nearest capable agent */`
7 $\quad$ **if** $a^* =$ null **then**
8 $\quad\quad K_{pending} \leftarrow K_{pending} \setminus \{k^*\}$ `/* discard the task */`
9 $\quad$ **else**
10 $\quad\quad t \leftarrow t_{a^*}^{free}$
11 $\quad\quad t^s \leftarrow t + \rho(l_{a^*}^t, l_{k^*})$
12 $\quad\quad t^c \leftarrow t^s + \lceil w_{k^*}/p_{a^*} \rceil$
13 $\quad\quad \Gamma^* \leftarrow \Gamma^* \cup \{\ \tau_{t^s,t^c}^{a^* \to k^*}\ \}$ `/* allocate the task */`
14 $\quad\quad K_{pending} \leftarrow K_{pending} \setminus \{k^*\}$
15 $\quad\quad t_{a^*}^{free} \leftarrow t^c$ `/* update the agent's availability time */`

Complexity Analysis. The EDF heuristic is $\Theta(m(m+n))$. Assuming the number of agents is limited (such as in search-and-rescue scenarios), the computational complexity of EDF heuristic can be reduced to $\Theta(m^2)$ (i.e., EDF is polynomial in the number of tasks).

5.2 Nearest Task First Heuristic

The Nearest Task First (NTF) heuristic, which is greedy too, prioritizes the task that requires travel the least (Algorithm 5). Specifically, it allocates to a capable agent that becomes available the soonest, the task which is the nearest. The heuristic selects tasks in a distance greedy fashion and agents in a time greedy manner. It iterates over the set of pending tasks, until they are all assigned or no agent can take more tasks (Algorithm 5). Each time, we select the agent that becomes ready the soonest (line 6), to assign it to the closest feasible task (line 8)—we arbitrarily pick up one agent if it happens that many become available simultaneously. If an agent can take no more task, it is removed from the set of potential agents to not be considered again (line 9,10). Prioritizing a task that has a nearby capable agent may reduce – at least in the short run – the agents's travel times; consequently, that increases their working times. Intuitively, selecting the earliest available agent balances the number of allocated tasks among the agents, while it helps to quickly ($\Theta(n)$) find an allocation.
Complexity Analysis. Similarly to EDF heuristic, NTF heuristic is $\Theta(m^2)$ (i.e., its runtime is quadratic in the number of tasks). However, the hidden factors suggest that NTF heuristic maybe, in practice, slower than EDF heuristic.

Algorithm 5: Nearest Task First (NTF) heuristic

Input: K, A
Output: Γ^*

1 $\Gamma^* \leftarrow \emptyset$
2 $K_{pending} \leftarrow K$
3 $A_{pot} \leftarrow A$ /* potential agents */
4 $t^{free} \leftarrow \langle 0, \ldots, 0 \rangle$
5 **while** $K_{pending} \neq \emptyset$ **and** $A_{pot} \neq \emptyset$ **do**
6 $\quad a^* \leftarrow \text{argmin}_{a \in A_{pot}} \{t_a^{free}\}$ /* earliest ready agent */
7 $\quad t \leftarrow t_{a^*}^{free}$
8 $\quad k^* \leftarrow \text{argmin}_{k \in K_{pending}} \{\rho(l_{a^*}^t, l_k) \mid t + \rho(l_{a^*}^t, l_k) + \lceil w_k / p_{a^*} \rceil \leq d_k\}$ /* a^*'s nearest feasible task */
9 $\quad$ **if** $k^* = NIL$ **then**
10 $\quad\quad A_{pot} \leftarrow A_{pot} \setminus \{a^*\}$ /* discard the agent */
11 $\quad$ **else**
12 $\quad\quad t^s \leftarrow t + \rho(l_{a^*}^t, l_{k^*})$
13 $\quad\quad t^c \leftarrow t^s + \lceil w_{k^*} / p_{a^*} \rceil$
14 $\quad\quad \Gamma^* \leftarrow \Gamma^* \cup \{ \tau_{t^s, t^c}^{a^* \rightarrow k^*} \}$ /* allocate the task */
15 $\quad\quad t_{a^*}^{free} \leftarrow t^c$
16 $\quad\quad K_{pending} \leftarrow K_{pending} \setminus \{k^*\}$

5.3 One Step Look-ahead Search

One Step Look-Ahead (LA1) search allocates tasks, step by step, based upon the near future consequences. At each step, LA1 selects the task assignment that allows a maximum number of tasks to be feasible in the next allocation step [23]. It is a tree search with pruning: an agent is selected based on the earliest completion first rule, whereas a task is selected based on maximum tasks reachability (Algorithm 6). First, we simulate the allocation of every pending task to its fittest agent (lines 5–14). To do so, we look for the agent a' that can complete the current task k the soonest (line 6). Then, we count how many tasks remain feasible if a' is allocated to k (line 10–13), which we refer to as k's degree of task accessibility, noted $degree_k$ (line 14). A task k' is accessible if it is feasible: to the current agent a', after honoring its assignment, or to any other agent (the first and second condition of line 13). The best task k^* has the highest value of degree of task accessibility (line 15). We restrict the look-ahead to one step, since looking further ahead significantly increases the computation overhead. LA1 increases the number of allocated tasks in the short-run, while it somewhat enables the agents to travel less; thus, to work more. Consequently, LA1 should increase the number of allocated tasks.

Complexity Analysis. LA1 is $\Theta(nm^2)$. Knowing that $n \leq m$ in MRTA-STC, a rough upper bound to LA1 is $\Theta(m^3)$. Since often $m \ggg n$; LA1 might be $\Theta(m^2)$, similarly to EDF and NTF heuristics. In practice the hidden factor may make the difference.

5.4 Insertion Based Heuristics

Insertion-based heuristics are classical approaches to routing and scheduling problems [26] which now come often embedded in advanced heuristics, such as metaheuristics [61, 62, 31].

Algorithm 6: One step Look-Ahead Search (LA1)

Input: K, A
Output: Γ^*

$\Gamma^* \leftarrow \emptyset$
$K_{pending} \leftarrow K$
$t^{free} \leftarrow \langle 0 \dots 0 \rangle$
repeat
  forall $k \in K_{pending}$ **do** `/* find k's fittest agent */`
    $a' \leftarrow \text{argmin}_{a \in A} \{t' = t + \rho(l_a^t, l_k) + \lceil w_k / p_a \rceil \mid t = t_a^{free}, t' \leq d_k\}$
    if $a' =$ null **then**
      $K_{pending} \leftarrow K_{pending} \setminus \{k\}$
    else `/* one step look-ahead */`
      $count \leftarrow 0$
      Computes t_k^s and t_k^c
      forall $k' \in K_{pending} \setminus \{k\}$ **do** `/* count tasks accessible after k */`
        if $(t_k^c + \rho(l_k, l_{k'}) + \lceil w_{k'} / p_{a'} \rceil \leq d_{k'})$ **or**
        $(\exists a \in A \setminus \{a'\} : t_a^{free} + \rho(l_a^{t_a^{free}}, l_{k'}) + \lceil w_{k'} / p_a \rceil \leq d_{k'})$ **then**
          $count \leftarrow count + 1$
      $degree_k \leftarrow count$
  $k^* \leftarrow \text{argmin}_{k \in K_{pending}} degree_k$
  $a^* \leftarrow$ fittest agent for k^* `/* line 6's formula where` $k = k^*, a' = a^*$ `*/`
  Compute $t_{k^*}^s$ and $t_{k^*}^c$
  $\Gamma^* \leftarrow \Gamma^* \cup \{ \tau_{t_{k^*}^s, t_{k^*}^c}^{a^* \rightarrow k^*} \}$ $t_{a^*}^{free} \leftarrow t_{k^*}^c$
  $K_{pending} \leftarrow K_{pending} \setminus \{k^*\}$
until $K_{pending} = \emptyset$

Basically, an insertion-based heuristic constructs routes – or tours – by inserting visits to nodes, one by one, based on some cost, such as the distance or the delay added to the tour. These route constructing heuristics differ on the number of routes considered during the insertion and the cost calculation used to select the visit to insert. We study some insertion-based heuristics on MRTA-STC: we show how we implement them for our problem and how we improved some of them. On one hand, we compare insertion-based heuristics so we can select which to embed in an advanced approach. On the other hand, we provide them for time-critical and computation power limited MRTA-STC settings. To so, we first state the assumptions and introduce the concepts needed.

5.4.1 Basic Definitions

Adopting the routing and scheduling terminology, we designate an agent's single assignment by a visit and an agent's sequence of assignments by a route (or tour). Let $route_a$ be agent a's route, v a visit to some task's node by some agent, and, specifically, $v_{k,a}$ agent a's visit to task k. Then, let $v_{k,a}^*$ be the best visit that agent a can currently make to task k. The best visit to task k, among all the agents' visits, is noted v_k^*. During a route construction, a task's visit maybe inserted at many places. Let $v_{k,a,j}$ designates the j^{th} visit in $route_a$ that can be made to task k (by agent a). Let $\{v_{k,a,j}\}_{j \in route_a}$ be the set of all visits that agent a can (currently) make to task k (i.e., all task k's visits that can be inserted in $route_a$).Inserting a visit v in a route requires to move forward (delay) the visits that follow. Let $shift_v$ be

that delay. Since the tasks have deadlines, before delaying a visit, we make sure that the visits that follow the visit to insert remain feasible. Let $maxshift_j$ be that affordable delay for visit j. Let v be a potential visit to insert and j ($j > v$) a visit that follows v on the same route. Visit v can be inserted if and only if $shift_v \leq maxshift_j, \forall j > v$. We denote by $v+1$ a visit that just follows v (on the same route).

5.5 Sequential Insertion Heuristic

The Sequential Insertion Heuristic (SIH) constructs routes one by one, by inserting task visits. The SIH's implementation we propose for our problem is a special case of Solomon's first insertion heuristic for VRPTW [26], where the insertion's delay is the visit's selection criterion. Our implementation is identical to the first heuristic that Labadi et al.'s embed in a metaheuristic for TOPTW [62], except that we use Vansteenwegen et al's. visit selection criterion, to speed up the search [61].

Algorithm 7: Sequential Insertion Heuristic (SIH)

Input: K, A
Output: Γ^*

1 $\Gamma^* \leftarrow \emptyset$
2 $K_{pending} \leftarrow K$
3 **forall** $a \in A$ **do** `/* fill routes one by one */`
4 **repeat**
5 **foreach** $k \in K_{pending}$ **do** `/* find a's best visit to k */`
6 $v^*_{k,a} \leftarrow \text{argmin}_{v \in \{v_{k,a,j}\}_{j \in route_a}} \{shift_v \mid shift_v \leq maxshift_{v+1}\}$
7 $v^*_a \leftarrow \text{argmin}_{v \in \{v^*_{k,a}\}_{k \in K_{pending}}} \{shift_v\}$ `/* the best visit in a's route */`
8 **if** $v^*_a \neq$ null **then**
9 insert v^*_a into Γ^*
10 $K_{pending} \leftarrow K_{pending} \setminus \{v^*_a$'s task$\}$
11 **until** $v^*_a =$ null **or** $K_{pending} = \emptyset$ `/* the agent can visit no more task */`
12 **if** $K_{pending} = \emptyset$ **then return**

SIH heuristic arbitrarily selects an agent to fill its route, repeatedly until all the tasks are visited or all the routes are full (Algorithm 7). A visit to a task is considered for insertion, if it is feasible and minimizes the delay introduced to the route, once inserted, without making infeasible any visit that follows (line 6)—we detail in Chapter 7 how to technically do that. To find the best visit to insert in the current agent's route (line 7), we first look for the best visit that the agent can make to every pending task (line 5,6). That is, we search in the agent's route when the current task can be visited while causing the least delay to the visits that follow (line 5).

Complexity Analysis. SIH is $\Theta(nm)$. When the agents are as many as the tasks ($n \approx m$), it is quadratic ($\Theta(m^2)$). In MRTA-STC, where $n \lll m$, SIH should be $\Theta(m)$.

5.6 Fast Parallel Insertion Heuristic

Instead of considering the routes separately, Parallel Insertion Heuristic (PIH) looks for the best visit over all. The routes are thus built concurrently. For brevity without compromising

Algorithm 8: Fast Parallel Insertion Heuristic (faPIH)

Input: K, A
Output: Γ^*

1 $\Gamma^* \leftarrow \emptyset$
2 $K_{pending} \leftarrow K$
3 $V_{best} \leftarrow \emptyset$
4 **forall** $k \in K_{pending}$ **do** `/* initialize the tasks' best visits */`
5 | **forall** $a \in A$ **do**
6 | | $v^*_{k,a} \leftarrow \text{argmin}_{v \in \{v_{k,a,j}\}_{j \in route_a}} \{shift_v \mid shift_v \leq maxshift_{v+1}\}$
7 | $v^*_k \leftarrow \text{argmin}_{v \in \{v^*_{k,a}\}_{a \in A}} shift_v$
8 | $V_{best} \leftarrow V_{best} \bigcup \{v^*_k\}$
9 **while** $K_{pending} \neq \emptyset$ **do** `/* build routes concurrently */`
10 | $v^* \leftarrow \text{argmin}_{v \in V_{best}} shift_v$
11 | insert v^* into Γ^*
12 | *updatedRoute* $\leftarrow v^*$'s route
13 | $K_{pending} \leftarrow K_{pending} \setminus \{v^*\text{'s task}\}$
14 | **forall** $v \in V_{best}$ **do** `/* update the tasks' best visits */`
15 | | **if** *v 's route* = *updatedRoute* **then**
16 | | | re-compute the best visit to v's task `/* idem to lines 5-8 */`

completeness, we expand on PIH later on (Chapter 7). Here, we rather show how we speed up PIH's implementation with fast Parallel Insertion Heuristic (faPIH) (Algorithm 8). Similarly to SIH, (fa)PIH (i.e., PIH and faPIH) insert task visits at different route positions, however, (fa)PIH construct the routes simultaneously (line 5–7). Moreover, faPIH ensures the visit to insert and those that follow (on the same route) are feasible (line 6). The visit with the lowest route shift is selected. Instead of re-computing the best visit in every route (as PIH), faPIH maintains a memory with the best visits to the pending tasks (line 4–8, 14–16). Likewise, we accelerate the selection of the visit to insert at each iteration (line 10). When a route is modified (i.e., after a visit insertion), it may no longer hold the best visits to some pending tasks. So, we update the best visit to a pending task if that best visit's route has been altered (line 15–16).

Complexity Analysis. In theory, faPIH is as slow as PIH ($\Theta(m^3)$), slower than SIH, but in practice faPIH should generally be faster than PIH; for it is very unlikely that all the pending tasks would need an update of their best visits at every iteration.

5.7 Parallel Insertion Heuristic with Task Selection

By minimizing a visit's insertion delay, PIH heuristic saves the agents' travel time and increases their work time. We propose a variant of PIH, noted xPIH, that uses the EDF heuristic to secure the allocation of urgent tasks, while it keeps the advantages of PIH—i.e., delay minimization and route load balancing through concurrent routes construction (Algorithm 9). Specifically, xPIH prioritizes the earliest deadline task (line 2) and the visit thereof that minimizes the insertion delay in all the routes (line 5). Thus, we decrease the search cost from PIH, for each task is considered for allocation only once. By incorporating EDF, we think we may increase the task allocation rate. Since the most constrained tasks' – with regard to deadline – feasibility is likely to drop quickly along the allocation process,

Algorithm 9: Task Selection Parallel Insertion Heuristic (xPIH)

Input: K, A
Output: Γ^*

$\Gamma^* \leftarrow \emptyset$
$K_{pending} \leftarrow K$ sorted in a non-decreasing order of tasks' deadlines
forall $k \in K_{pending}$ **do** `/* allocate the earliest deadline task first */`
  forall $a \in A$ **do** `/* find the task's low-cost visits in the routes */`
    $v^*_{k,a} \leftarrow \text{argmin}_{v \in \{v_{k,a,j}\}_{j \in route_a}} \{shift_v \mid shift_v \leq maxshift_{v+1}\}$
  $v^*_k \leftarrow \text{argmin}_{v \in \{v^*_{k,a}\}_{a \in A}} shift_v$ `/* the task's best visit */`
  if $v^*_k \neq$ null **then** insert v^*_k into Γ^*

making so such tasks harder and harder to visit and delaying the routes much more.

Complexity Analysis. xPIH is at worst $\Theta(m^2)$. So it is theoretically slower than SIH, but faster than faPIH and PIH. We think that, in practice, on the average, xPIH scales even better because of a shorter runtime $T(n, m) = \Theta(m^2 - mn)$.

5.8 Empirical Evaluation

5.8.1 Experimental Setup

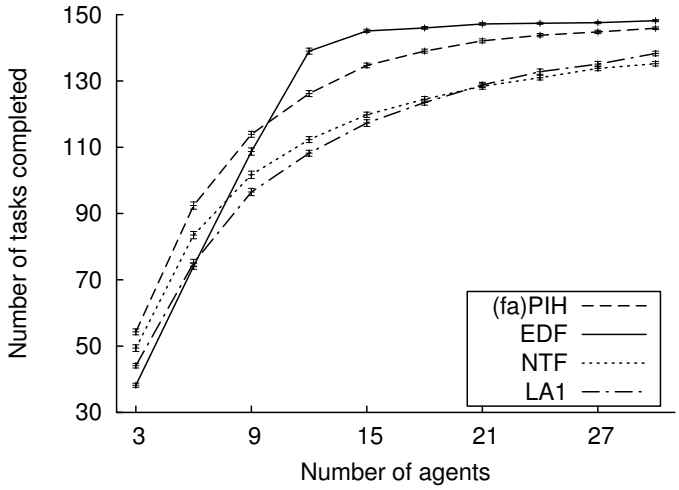
To evaluate the heuristics we proposed for MRTA-STC, we coded them in Java and we sampled some test instances as follows. We randomly scatter the agents and the tasks on a 120×120 grid map with the uniform distribution (i.e., $l^0_a, l_k \sim (U(0, 120), U(0, 120))$). We calculate the distances that the agents travel with the "Manhattan distance" formula. We set the agents' velocity at a constant unitary value; therefore, the time an agent takes to travel between two tasks is equivalent to the distance between them. We draw the agents' work capacities at random such that they range from one to three times the unitary capacity value (i.e., $p_a \sim U(1.0, 3.0)$). To get tasks' workloads' and deadlines' values, we uniformly sample over large intervals (to simulate widely different scenarios) as: $d_k \sim U(30, 600)$ and $w_k \sim U(10, 60)$.

We fix the number of tasks at 150 and we vary the number of agents from 3 to 30, by increments of 3. For each problem size, we run the heuristics on 100 different instances sampled with the same configuration (as shown above). The testset gathers ten problems with different size that sum up to 1000 instances [1]. We run the experiments on an Intel(R) Core(TM) i7 CPU, clocked at 2.20 GHz, with 8 Giga of RAM, under Linux.

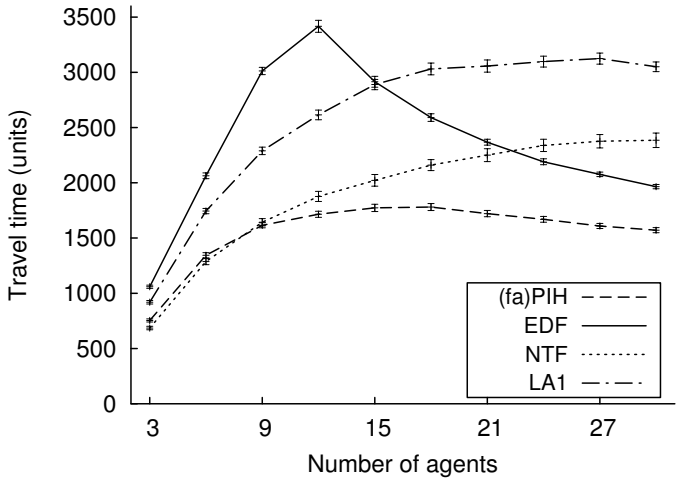
5.8.2 Results

We report the means of solution qualities and travel costs per problem set (Figures 5.1 and 5.2) and we plot the 95% confidence interval as an error bar for each problem set (100 instances). In Table 5.1 and Figure 5.3, we report the mean (empirical) computation times. The heuristics (fa)PIH constantly perform better than NTF and LA1 heuristics (Figure 5.1), namely, (fa)PIH allocate more tasks while they generate routes that require

[1] available online at: http://tinyurl.com/ts1tastp0



(a) Solution Quality



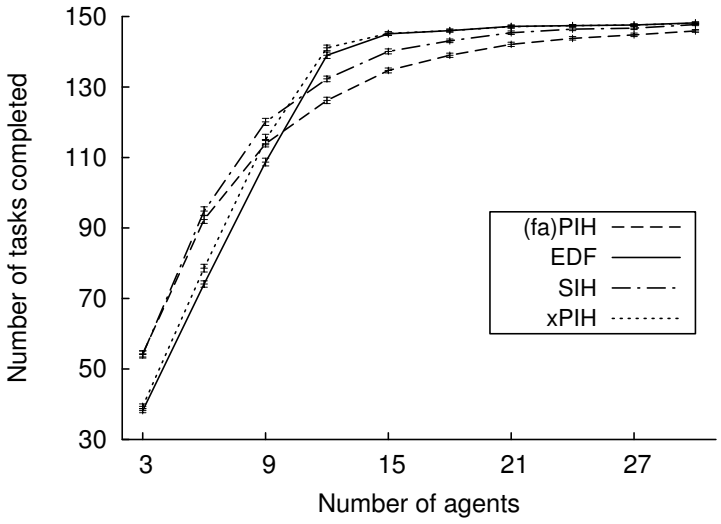
(b) Travel Cost

Figure 5.1: Basic heuristics performance results on MRTA-STC (part 1)

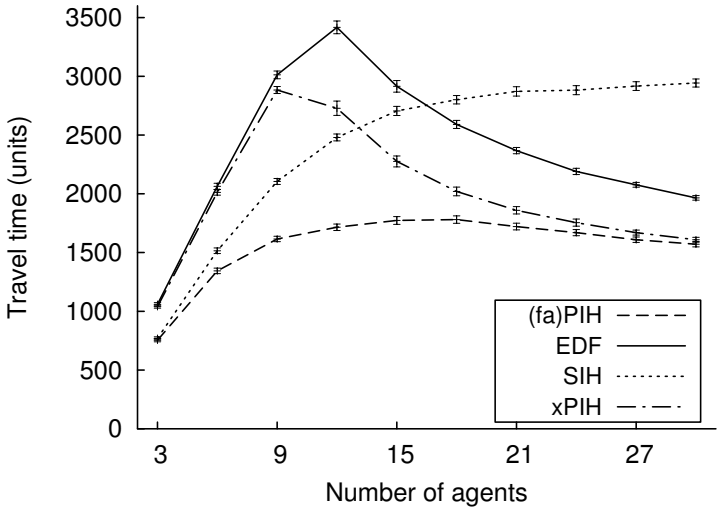
less travel time to visit the tasks. We confirm that faPIH is faster then PIH, it is noticeably faster than LA1 and a little slower than NTF (Table 5.1).

The heuristic faPIH is comparable to EDF with regard to solution quality and faPIH performance gap tends to shrink as more agents become available. Surprisingly, EDF quickly tends to allocate almost all the tasks, as more agents are available [2]. Moreover, EDF is remarkably faster than faPIH (Table 5.1). Nonetheless, faPIH better allocates tasks on difficult problem (fewer agents), it improves PIH's response time up to 94% and it saves the travel distance cost compared to EDF (Figure 5.1b). SIH generally allocates tasks more than faPIH does, particularly when there is a moderate number of agents (Figure 5.1a). xPIH is almost identical to EDF with regard to solution quality and response time—it slightly better allocates tasks when there are few agents. As the number of agents increases (above nine), xPIH allocates more tasks and reduces the travel cost compared to SIH. xPIH is clearly faster than faPIH and SIH (Figure 5.3). The results are statistically significant for the error bars generally do not overlap. We conclude that SIH offers the best solution

[2] That made us suspect the convenience, per chance, of some of the testset instances for EDF



(a) Solution Quality



(b) Travel Cost

Figure 5.2: Basic heuristics performance results on MRTA-STC (part 2)

Table 5.1: The Basic Heuristics computation time (ms) - part 1

n	3	6	9	12	15	18	21	24	27	30
EDF	0.2	0.1	0.1	0.1	0.2	0.1	0.2	0.2	0.1	0.2
NTF	2.1	1.2	1.1	1.1	1.3	1.2	1.3	1.4	1.3	1.3
LA1	21.6	37.3	51.3	60.3	71.0	79.5	92.2	89.8	94.7	98.1
PIH	8.3	18.9	29.1	35.9	41.2	44.3	48.3	51.6	54.9	58.2
faPIH	3.2	3.6	3.7	3.5	3.5	3.3	3.3	3.3	3.2	3.1

quality within few seconds when the number of agents is low. As more agents become available, xPIH achieves the best solution quality with a travel cost and response time that are competitive compared to other heuristics. Yet, faPIH and SIH performances are more stable. Consequently, we think that faPIH or SIH maybe the suitable heuristic to embed in a multi-restart type metaheuristic. The remaining heuristics maybe used in some advanced heuristic that combines them, somehow, to achieve a better performance. In computation power restricted settings, NTF is suitable for problem instances with a few to a moderate

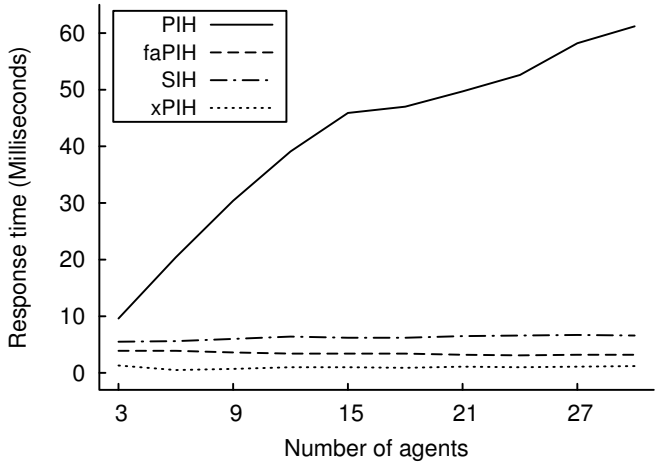


Figure 5.3: Basic heuristics computation time on MRTA-STC (part 2)

number of agents (with some trade-off of solution quality). Whereas EDF is suitable for a large number of agents.

5.9 Conclusion

We have proposed fast and easy-to-implement approaches to our multi-robot task allocation problem, which improve solution qualities and response times compared to the market-based approach (Chapter 4). These are heuristics that insert visits to tasks, route by route or simultaneously, at the end or anywhere in the route, by minimization the travel distance, the completion time, the insertion delay, or by maximizing tasks reachability or prioritizing urgent tasks. We have showed how to implement the heuristics, we analyzed their computational complexities and we experimentally compared them on some more elaborated MRTA-STC instances. We found that the insertion-based heuristics perform generally better. We show how to improve the response time of the parallel insertion heuristic and how to improve the solution quality of the earliest deadline first heuristic. The next chapters build upon the heuristics studied here to yield higher solution quality, while keeping the response time acceptable.

Chapter 6

On the Tuning and Evaluation of Iterated Local Search

Iterated local search (ILS) is a metaheuristic that proved to be effective on many intractable combinatorial optimization problems. As most metaheuristics, ILS is likely to require a good deal of tuning before it can achieve satisfactory results. Generally, tuning approaches do not consider the specificities of the metaheuristic under development. We want to tune ILS based on its main feature: "search guidance". To do so, given the local search procedure (or the embedded heuristic), we suggest to tune ILS while comparing against two baselines: multi-restart local search (MRLS), where there is no "search guidance", and randomized iterated local search (rILS), where the search guidance is random to some extent. By so doing, we can tell if the perturbation and/or the acceptance criterion (ILS guidance modules) should be tweaked further. In Section 6.1, we briefly review metaheuristic tuning and evaluation. In Section 6.2, we give ILS's general scheme and how ILS works. In Section 6.3, we review a competitive ILS for TOPTW and show how we adapt it to MRTA-STC problem. In Section 6.4, we show how to derive two basic metaheuristics, given the embedded heuristic, so to quickly evaluate ILS, eventually, to tell if there is room for tuning the search guidance, specifically. In Section 6.5, we experimentally demonstrate that the studied ILS lacks tuning, since it failed to outperform the trivial baselines.

6.1 Introduction

Metaheuristics are high level heuristics that proved to be effective to tackle many hard combinatorial optimization problems. Results in the literature indicate that metaheuristics are among the state-of-the-art to approach problems for which we do not know efficient algorithms [31, 63]. Again, metaheuristics are appealing because they are algorithmic templates that can be easily applied to practically any optimization problem. To become functional, a metaheuristic needs configuration: some modules require instantiation and some parameters – numerical or qualitative – need to be set. A quick-and-dirty configuration (or tuning) may outcome fairly good results. Nevertheless, an extra tuning is necessary for the metaheuristic to become competitive or the state-of-the-art. Unfortunately, tuning is not straightforward because of the lack of theoretical analysis tools [60]. So typically, metaheuristics are rather analyzed and tunned experimentally on established benchmarks.

The tuning is intertwined with the design and implementation. Tuning aims to improve the metaheuristic's modules individually and/or in combination, while fine-tuning seeks to further improve it but to a less extent, by looking for the search configuration parameters (free parameters) values that yield the best (or satisfactory) results [1]. For instance, let us assume a metaheuristic M with three modules: m_1, m_2 and m_3, a baseline heuristic B (or the set of optimal solutions O, when available) and a performance metric c. Let m_{ij} designates an instance j of module $m_i \in M (i = 1 \dots 3)$. Let m_i^* be module m_i's best instance—indeed, it is rather the module's best-found instantiation, since we may not be able to find the best one nor to prove it so otherwise. If m_i is the LS in a SA, for example, we can have m_i^* by looking for many m_i and comparing SA's performance with B's (or with O). The combination $\{m_1^*, m_2^*, m_3^*\}$, a metaheuristic M's instance that is obtained through tuning the modules individually, may not be the best. Consequently, we need to consider tuning a metaheuristic's module in conjunction with the others, as well. Indeed, the modules are inter-dependent and there interaction may not be well understood. There is anecdotal evidence that about 10% of the total time dedicated to design and test a new metaheuristic (or a heuristic) is spent in development (including tuning), while the remaining 90% is spent in fine-tuning [18]. Nonetheless, we think tuning a metaheuristic is as important as fine-tuning, maybe even more; since, intuitively, the latter cannot compensate for underachieving modules and/or a bad combination thereof. Usually, metaheuristics are tuned by hand in a trial-and-error fashion, guided by some rules of thumb [8]. The tuning can be specific to the metaheuristic at hand or based on problem-specific knowledge. If the metaheuristic is modular—usually, it is the case such as SA and TS—, it is tuned component per component, then by considering the components all together [12]. Some authors adopt a descriptive analysis based on fractional design [16]. For example, Xu and Kelly attempt to identify the relative contribution of five components in TS [17]: they disable each component, one at a time, execute the resulting algorithm on seven instances and compare the results to draw conclusions on the effectiveness of each component. The designer may instead selects a module instance that experimentally proved to be successful or that is supported by a theoretical proof. For instance, Hajek gives a parametric cooling temperature schedule that guarantees the convergence of SA to a global optimum, under some condition on the parameter [65]. Additionally, Boese et al. experimentally show that periodic (or warming) temperature schedules, in SA, can lead to a better performance in time-bounded applications [66].

Iterated Local Search (ILS) is a metaheuristic which analysis and tuning are not straightforward. In essence, ILS is a repetitive LS run, where a run's output is partially used as an input in the next run, in a way that increases the likelihood of improving the currently best-found solution. ILS performance depends on the choices made on its four components: the initial solution, the LS (or the embedded heuristic), the acceptance criterion and the perturbation procedure. We refer to the last two by the "search guidance" modules. ILS performance depends also on the components combination. Lourenço et al. suggest an individual tuning of ILS's modules, before performing a global tuning. The

[1] It is usual in the literature to rather refer with "tuning" to "fine-tuning". Here, we instead adopt Birattari's terminology to distinguish between improving the metaheuristic's modules *vs.* improving the free parameters setting [64]

latter process can be approximated by successively tuning each component, while letting the remaining components fixed, until no improvement can be found on any component [12]. However, this approach is time consuming: many components' combinations are likely to be tested plus fine-tuning each combination.

In what follows, we study an ILS on MRTA-STC problem, so to build on top of it based on tuning the search guidance. To do so, we check how well ILS is tuned. Our hypothesis is that we can achieve fair results if we restrict the tuning to the "search guidance" modules. This is motivated by the fact that: a good heuristic to embed is often available, the initial solution has generally little impact—particularly when the search time is long—and the search guidance modules are the main ILS's feature. The contributions of this chapter are: (1) two trivial metaheuristic baselines for ILS that are easy to derive, given the LS and a little information about the problem, consequently (2) a global tuning of ILS directed toward the search guidance modules, (3) a case-study implementation of the baselines on our problem and the ILS evaluation. The first baseline we suggest, though not novel, a few literature works consider to evaluate against it and give details on of its implementation. The second baseline is new and specific to ILS.

6.2 Iterated Local Search

Let S be the search space and S^* the space of local optima. Local search (LS) maps a solution from S to S^*. ILS, however, maps multiple solutions from S^* to S^*, by repeatedly doing a LS that is seeded with the current local optimum. Likewise, ILS escapes local optima and attempts to find better ones, or a global optimum. It explores S^* more efficiently than a multi-restart local search (MRLS), which repeatedly maps solutions from S to S^*, without exploiting the search history. Contrary to MRLS, which is a "random walk" in the space of local optima (S^*), we see ILS as a "guided walk" in S^*.

Algorithm 10: Iterated local search architecture

Input: *problem, search configuration parameters, initial solutions*$_0$, *termination condition*
Output: s^*

```
1 s ← LocalSearch(s_0)
2 s* ← s
3 repeat
4 |   s' ← Perturbation(s)
5 |   s*' ← LocalSearch(s')
6 |   s ← AcceptanceCriterion(s, s*')
7 |   if f(s*') > f(s) then s* ← s*'
8 until termination condition
```

The search starts from an initial solution s_0 which is generally empty, random or the output of some fast greedy heuristic (Algorithm 10). Then, it iterates until a given termination condition is met. At each iteration, ILS tries to improve the best-so-far solution s^* by modifying the current solution s and applying a LS. The modification (or perturbation) procedure changes s to some extent, depending on a perturbation strength which is either given by the user (i.e., a free parameter) or coded by the designer. The perturbation's

outcome, s', is a partial solution—i.e., we can add solution elements to s'. By applying a LS on s', we obtain a new local optimum $s^{*\prime}$ that maybe better then the current or the best-so-far solution (line 6,7, Algorithm 10, where $f : S \to \mathbb{R}$ is the solution quality function, to maximize). However, when a new solution, $s^{*\prime}$, is less valuable then the current one, s, the acceptance criterion module decides whether to carry on the search from s or $s^{*\prime}$. The perturbation and the acceptance criterion modules altogether define the search trajectory of ILS. Applying weak perturbations along with only accepting better solutions encourages the search intensification. Inversely, if we perturb the solutions significantly and accept them easily, we then encourage search diversification. Some balance between the two search tendencies is recommended or the best ratio maybe found experimentally.

6.3 Iterated Local Search for MRTA-STC

Vansteenwegen et al.'s ILS is a fast, simple and competitive approach to TOPTW. It has a small score gap (1.8% on average) compared to the best-known, on a large set of benchmarks [61]. A recent comparison of the state-of-the-art approaches to the TOPTW—which takes into account the heterogeneity of the computational platforms— reports higher solution qualities (then ILS's) obtained by ACO, SA and hybrid local search [63]. Yet, Vansteenwegen et al.'s ILS – noted ILS from now on – keeps relatively a small percentage of score deviation from the best-known, while it globally shows the promptest response. Therefore, we opted for ILS as a basis to propose and evaluate a better approach to MRTA-STC. To do so, we first study the ILS on MRTA-STC through an evaluation against trivial metaheuristics. Before doing so, we briefly present the design choices made in ILS for TOPTW and project them on our problem.

The LocalSearch procedure in ILS (Algorithm 10) is rather instantiated with an insertion-based heuristic, namely PIH (Chapter 5). We recall that PIH considers all the tasks, the tours and tour positions when constructing the routes. It prioritize a visit v based its profit, Π_v, and the delay, shift_v, it adds to the tour [61]. ILS computes the desirability of inserting visit v as:

$$\frac{\Pi_v^2}{\text{shift}_v} \tag{6.1}$$

Thus, the embedded heuristic gives importance to profit gain more then to delay minimization when considering the visit to insert. In MRTA-STC, the tasks have the same priority, similarly to the special case TOPTW (Chapter 3), then we substitute Formula 6.1 with:

$$\frac{1}{\text{shift}_v} \tag{6.2}$$

Likewise, we only minimize the delay added to the tour, to eventually enable further visits to be inserted so on and so forth. We compute shift_v similarly to ILS for TOPTW, except that there is no waiting time at a task's location, the deadline restricts the time by which a task should be finished—instead of the time by which the task can be started—and the service time depends on the capacity of the serving agent (c.f., see Chapter 3). The initial solution is empty (Algorithm 10). The termination condition holds after the given number,

$maxTrials$, of trials have been done without improving the solution. The perturbation phase has a variable perturbation strength r and a variable perturbation start x. The user should configure $maxPerturbationCoef$, "maximum perturbation strength coefficient" parameter to bound the perturbation's strength. The perturbation removes a sequence of r visits from each tour, starting at different positions x_{tour}, at each iteration. The perturbation strength r is gradually incremented along; it is re-initialized with the unitary value when a new best-so-far solution is found, or when some threshold, which is partially determined by $maxPerturbationCoef$, is reached [61].

While ILS is among the state-of-the-art approaches to TOPTW, we cannot conclude how efficient is its search guidance, unless we measure it. We suggested to do so, on MRTA-STC, through two baselines which are easy to implement and tune.

6.4 Trivial Baselines for Iterated local search

The strength of ILS lies in a biased sampling from the set of local optima [12]. Given the LS procedure, the sampling efficiency depends on the instantiations of the perturbation and the acceptance criterion modules, combined. Interestingly, even with a quick instantiation of the search guidance modules, ILS performs better than many LS which are randomly seeded [12]. Consequently, a fairly tuned ILS should perform, at least, better than: a multi-restart local search and an ILS implementation with an arbitrary search guidance. Given the embedded heuristic (or a LS procedure), comparing ILS against the two suggested trivial baselines helps, in particular, to evaluate ILS's search guidance.

6.4.1 Multi-restart Local Search

Algorithm 11: Multi-restart local search (MRLS)

Input: K, $nbrRestarts$, $nbrRandomSolutionElementRatio$.
Output: solution s^*

$s^* \leftarrow \emptyset$
$nbrRandomSteps \leftarrow nbrRandomSolutionElementRatio \times f(\text{LocalSearch}(\emptyset))$
repeat
  $K_{pending} \leftarrow K$
  $s_0 \leftarrow \emptyset$
  repeat `/* seed the LS randomly */`
    $v \leftarrow$ a random visit to an arbitrary task $k \in K_{pending}$
    insert v into s_0
    $K_{pending} \leftarrow K_{pending} \setminus \{v$'s task$\}$
  until $nbrRandomSteps$
  $s \leftarrow \text{LocalSearch}(s_0)$
  if $f(s) > f(s^*)$ **then** $s^* \leftarrow s$
until $nbrRestarts$

The simplest way to improve a LS outcome's quality is to repeat the LS many times with different seeds. Typically, the seeds (initial solutions) are arbitrary; which makes the search a sequence of independent random samplings in S^*. Thus it is referred to as: "random restart" search or Multi Restart Local Search (MRLS). The rational of MRLS is that if (i)

the fraction of satisfactory local optima[2] is sufficiently large and if (ii) these local optima are uniformly distributed in the search space, by repeating the LS procedure a sufficiently large number of times, the probability of hitting a satisfactory local optimum is then quite high [12]. However, neither of these assumptions is justified in many applications. Furthermore, as the problem instance size increases, MRLS has a lower and lower probability of finding satisfactory local optima. YEt, its triviality makes it a good performance yardstick that any well-designed experimental analysis should include [64].

We instantiate MRLS for MRTA-STC by Algorithm 11. We keep the ILS's LocalSearch. We get the initial solution, s_0, by inserting some solution elements randomly. Specifically, we insert a number of visits which the tour, the task and the insertion position are selected arbitrarily (line 7). The number of random insertions, *nbrRandomSteps*, is s_0's size. Intuitively, it should be small to give room for enough subsequent heuristic – rational – insertions and somehow ensure to get fairly good solutions. We set *nbrRandomSteps*'s value, at runtime, based on the free parameter *nbrRandomSolutionElementRatio* which tells the ratio of random visits to the total number of visits in an ordinary LS solution (line 2, Algorithm 11). We empirically found that the best value for *nbrRandomSolutionElementRatio* should be very small, in our case study. In each of the *nbrRestarts* iterations, MRLS initializes the LS with some arbitrary partial solution s_0 (line 6–10), does a LS and eventually updates the best-found solution s^* (line 11–12). The random sampling of s_0 allows the LS to hit different local optima. Ideally, *nbrRestart* should be set such that to hit as much different s^* as the time budget allows.

6.4.2 Randomized Iterated Local Search

Algorithm 12: Randomized iterated local search (rILS)

Input: A, K, $maxIter$, P
Output: s^*

```
1  s ← LocalSearch(∅)
2  s* ← s
3  iter ← 1
4  repeat
5  |  s' ← Perturbation(s)
6  |  s*' ← LocalSearch(s')
7  |  s ← s*' with probability P
8  |  s ← s with probability 1 − P
9  |  if f(s*') > f(s) then s* ← s*'
10 |  iter ← iter + 1
11 until iter = maxIter or all k ∈ K are visited
```

Contrary to MRLS, ILS is guided by the perturbation and acceptance criterion modules, that enables it to do better then a mere random restart of LS. It is therefore important to tune these modules before comparing ILS against any elaborated approach. Given the ILS's embedded heuristic, the termination condition and the initial solution, we derive a

[2]Local optima are satisfactory if their qualities are sufficiently close to a global optimum's quality, or they are rather so as being estimated by the practitioner on the target application.

randomized Iterated Local Search (rILS), where we only modify the perturbation and the acceptance criterion. If rILS performs better then ILS, we can conclude a shortcoming in ILS search guidance and suggest a tuning therein. The acceptance criterion is controlled by the probability P (Algorithm 12) We keep ILS's "random walk" acceptance criterion [61], by setting $P = 1.0$ (lines 7–8). As such, we ever carry on the search from the current solution, independently of the solution's quality. Hence, we encourage a exploration the most, similarly to MRLS, and do not exploit the search history at all. Alternatively, one can set the acceptance probability as $P = 0.5$ to accept or reject a solution uniformly at random, if we don't know whether more exploration or exploitation is better. Having an idea about the solution landscape—the case of some well-studied problems, as the TSP—, we may set the acceptance probability such that to disadvantage rILS and get an even lower performance bound for ILS. For instance, when the solution landscape features a few regions with high quality local optima, by setting P such as to encourage exploration (i.e., $P \in (0.5, 1.0]$), rILS is more likely to perform shorter then any tuned ILS.

Algorithm 13: Perturbation step of the randomized iterated local search

Input: $s, K_{pending}$
Output: s'

```
1 atl ← mean_{tour∈s} {length(tour)}
2 r ← a random number ∈ [1, atl]
3 s' ← s
4 forall tour ∈ s' do
5 |   if tour ≠ ∅ then
6 |   |   x ← a random number ∈ [0, length(tour) − 1]
7 |   |   Remove r sequential visits from tour starting from x
8 |   |   K_pending ← K_pending ∪ { the r removed visits' tasks}
```

We perturb the current solution s by removing an arbitrary number r of sequential visits from each tour (line 7, Algorithm 13). The position x where we start the removal is arbitrary too (line 6). As the tours have different lengths, we set the maximum number of visits to remove (from s) equal to the average tour length, atl (line 1). If the perturbation is longer than the tour, we remove all the visits. The removed visits' tasks are re-inserted into the set of unallocated tasks $K_{pending}$, so that they can be re-considered for allocation again. The rILS's perturbation phase is diverse as the perturbation strength and start are set randomly. Likewise, we enable search intensification and diversification, but in an arbitrary manner. Introducing randomness in the perturbation is also a trick to avoid re-cycling; a common issue in trajectory-based deterministic searches (as ILS). Thus, rILS is another good baseline, possibly better then MRLS, to quickly measure the effectiveness of "search guidance" in ILS.

6.5 Empirical Evaluation

6.5.1 Test Instances

We generate two groups of problem instances. In group 1, the instances admit solutions which tours have a small to a moderate length. In group 2, the instances admit solutions which tours are relatively long. The number of tasks is 100 over all, while the number of

agents varies as: $2, 3, 5$ and 7. The map is a 100×100 grid. The agents' and task locations are drawn uniformly at random. The travel velocity is commonly set at a unitary value. An agent a's work capacity $p_a \in \{1, 2\}$, it is set such that about one third of the agents would be twice more efficient then the rest. The tasks' workloads are set at 20. A task k's deadline, d_k, is normally drawn from $[d_k^{low}, d^{high}]$ for $25\%, 50\%, 75\%$ or 100% of the tasks. The remaining tasks take d^{high}, the slackest deadline. The tightest deadline a task k may take, d_k^{low}, is set such that k could be feasible to any agent that attempts it at the beginning of its tour. The value d^{high} is set such that if given: d^{high} is the tasks' deadline, 10 agents, a travel distance between any pair of tasks that is equal to one fourth of the longest travel distance (25 units); then the agents could visit all the tasks. We experimentally found that setting d^{high} likewise enables long tours (group 2's instances), whereas we enable relatively short tours (group 1's instances) if we set d^{high} at half of that setting, .

By combining the number of agents available with the percentage of normally sampled deadlines, we get 16 different problem configurations per group. We sample three instances per problem configuration to get 96 MRTA-STC instances [3]. A problem instance's name is coded as "r*gp*a*n*" followed by the instance number, when the instance pertains to group g ($g \in \{1, 2\}$), has a ratio $p/4$ ($p \in \{1 \dots 4\}$) of tasks with normally sampled deadlines (as shown above) and n agents available. For example, the problem instance named r23a501, is in group 2, has 3/4 tasks with normally distributed deadlines, 5 agents available and it is the first sampled with that configuration.

6.5.2 Expirement Setup

We implemented ILS, MRLS and rILS in Java. We ran the metaheuristics on an Intel i7 CPU clocked at 2.20 GHz with 8 Giga of short-term memory, under Linux operating system. We fine-tuned ILS by looking for the best search configuration parameters' values on a per problem group basis. We tested many values combinations, until we could not fairly improve the solution quality, without significantly increasing the computation time (which did not go beyond 15 seconds). The best configuration we have found (for ILS) is: maxTrials= 600, maxPerturbationCoef= 1/2 for group 1's instances, maxTrials= 600, maxPerturbationCoef= 2/5 for group 2's instances. We fine-tuned MRLS in a similar way. We did not need to fine-tune rILS. We made sure for the trivial baselines that the runtime does not exceed ILS's runtime. We ran ILS once, we ran MRLS and rILS 11 times each on every problem instance. Then, we computed the median, best and worst score gaps (in percentage) between ILS and the baselines (Equation 6.3). We also measure the computation time gaps (in seconds) between ILS and the trivial baselines in the median, best and worst runs of these latter (Equation 6.4).

$$\text{Score Gap} = \frac{\text{baseline score} - \text{ILS score}}{\max(\text{baseline score}, \text{ILS score})} \times 100\% \tag{6.3}$$

$$\text{Time Gap} = \text{baseline cutoff-time} - \text{ILS runtime} \tag{6.4}$$

Again, score is the number of allocated tasks (i.e., the solution quality). We redefine the "score gap" here as the percentage of deviation of one approach's score from the others',

[3] The test instances are available on the Web at http://tinyurl.com/taptc15in

such that positive values account for a shortcoming in ILS (Equation 6.3). We interrupt the baselines' run as soon as they hit their best solutions, let cutoff-time be that time. The computation "time gap" is the difference between the baseline's cutoff-time and ILS's runtime (6.4). Thus, negative values of the time gap indicate how longer, then the baseline, ILS was ran, eventually to catch up with (Equation 6.4).

We report the results of the experiments averaged per problem (Tables 6.1,6.2). ILS's score is comparable to MRLS's, even-though ILS is ran longer (Table 6.1). Specifically, ILS barely improves MRLS's median score on average (up to 2.59%) and only on less than half of the problems. ILS performs worse then the second baseline rILS. Specifically, in the median run, rILS achieves scores that are higher then ILS's (up to 4.15%) on most problems, within faster response times (Table 6.2). Interestingly, even if we consider the worst run, rILS does on average better than ILS on group 1's problems. In the median run, rILS outperforms or ties with ILS on 14 and 12 (out of the 16) problems of group 1 and 2, respectively. At the best run, rILS hits solutions with quality that is higher then ILS's on 30 (out the 32 problems, up to 5.05%), within a shorter response time. We further checked rILS is of a superior to ILS, by running rILS 100 times and plotting the distribution of the score. Typically, rILS always achieves a better score then ILS's: the score distribution is generally similar to that we have found on pb7 instance (Figure 6.1).

The superior performance of rILS can only be the result of the perturbation, since we kept the remaining ILS modules. Specifically, rILS's perturbation maybe diverse to an extent that enabled the search to better explore the solution space. Otherwise, the comparability of ILS performance with MRLS' and its under-performance compared to rILS, is a hint that ILS may have been trapped in search cycles (likely, for ILS is deterministic) and/or that it has a poor "search intensification" (a behavior absent in MRLS). Furthermore, the under-performance of ILS maybe linked to the way the search intensification and diversification are balanced. In Contrast, we think the stochasticity injected in rILS's

Table 6.1: ILS *vs* MRLS—ILS is comparable to the baseline

Input	Score Gap (%)			Time Gap (s)			Input	Score Gap (%)			Time Gap (s)		
	median	best	worst	median	best	worst		median	best	worst	median	best	worst
pb1	-1.30	-1.30	-1.30	-1.1	-1.1	-1.1	pb17	+0.66	+0.66	0.00	-2.1	-1.8	-1.9
pb2	+1.36	+1.36	+1.36	-0.9	-0.9	-0.9	pb18	+0.94	+1.63	0.00	-1.9	-1.8	-2.5
pb3	0.00	0.00	0.00	-0.8	-0.8	-0.8	pb19	0.00	+0.98	0.00	-2.1	-2.1	-2.2
pb4	+2.03	+3.50	+2.03	-0.7	-0.6	-0.7	pb20	+2.92	+2.92	+2.92	-1.4	-1.5	-1.4
pb5	+1.22	+1.22	+1.22	-1.2	-1.3	-1.4	pb21	-1.58	-0.47	-2.05	-2.3	-2.1	-2.7
pb6	+0.97	+0.97	+0.97	-1.6	-1.5	-1.6	pb22	0.00	+0.96	-0.65	-2.2	-2.3	-3.1
pb7	0.00	0.00	0.00	-1.5	-1.5	-1.5	pb23	-1.03	+0.68	-1.72	-3.2	-2.5	-3.3
pb8	+2.59	+2.59	+2.59	-1.3	-1.3	-1.3	pb24	-1.99	-0.60	-2.58	-3.6	-3.2	-3.8
pb9	0.00	0.00	-1.91	-0.7	-1.5	-2.9	pb25	0.00	+0.62	-1.04	+0.1	+2.5	-2.0
pb10	-0.57	-0.57	-0.57	-2.8	-2.9	-3.6	pb26	0.00	+1.08	-0.76	-2.2	+1.0	-2.1
pb11	-1.19	-1.19	-1.99	-3.0	-3.0	-2.2	pb27	-2.03	-0.48	-2.39	-2.3	-1.1	-2.9
pb12	+0.94	+0.94	0.00	-2.2	-2.1	-2.2	pb28	-1.70	-0.79	-1.70	-2.9	-2.2	-3.0
pb13	-2.22	-1.30	-3.13	-2.0	-2.9	-4.7	pb29	0.00	0.00	0.00	0.0	0.0	0.0
pb14	-2.31	-1.90	-3.26	-3.8	-1.1	-2.9	pb30	0.00	0.00	0.00	0.0	0.0	0.0
pb15	-2.77	-1.80	-3.60	-3.8	-2.4	-4.0	pb31	0.00	0.00	-0.30	+0.5	+0.8	0.0
pb16	-2.49	-0.93	-3.58	-4.4	-2.4	-4.9	pb32	-0.31	+0.72	-1.04	+0.4	+1.4	+0.7
Avg.	-0.25	+0.11	-0.74	-2.1	-1.8	-2.5	*Avg.*	-0.27	+0.53	-0.75	-1.7	-1.0	-2.0

Table 6.2: ILS *vs* rILS—ILS did worse than the baseline

Input	Score Gap (%)			Time Gap (s)			Input	Score Gap (%)			Time Gap (s)		
	median	best	worst	median	best	worst		median	best	worst	median	best	worst
pb1	0.00	0.00	0.00	-1.1	-1.1	-1.1	pb17	+0.66	+1.53	-0.67	-2.2	-1.6	-2.3
pb2	+1.36	+2.69	0.00	-0.9	-0.9	-0.9	pb18	+3.20	+3.20	0.00	-2.3	-2.0	-2.5
pb3	+1.41	+1.41	0.00	-0.7	-0.7	-0.8	pb19	0.00	+0.98	-1.49	-1.5	-1.6	-1.8
pb4	+3.50	+4.93	+3.50	-0.7	-0.7	-0.7	pb20	+4.03	+4.86	+2.06	-1.3	-1.2	-1.4
pb5	+4.15	+4.15	+3.00	-1.6	-1.6	-1.5	pb21	0.00	+1.09	-0.95	-2.7	-1.7	-2.6
pb6	+4.06	+4.06	+1.92	-1.5	-1.4	-1.4	pb22	+0.96	+1.59	-1.13	-2.6	-1.1	-3.2
pb7	+3.15	+4.95	+0.97	-1.4	-0.9	-1.5	pb23	-1.03	-1.03	-2.74	-2.5	-2.7	-3.5
pb8	+3.66	+5.05	+2.59	-1.0	-1.1	-1.1	pb24	-0.60	-0.60	-2.58	-2.9	-3.8	-3.9
pb9	+1.88	+2.61	+0.76	-2.6	-2.4	-3.0	pb25	+0.62	+1.64	0.00	-1.5	-0.6	-1.5
pb10	+1.32	+1.88	0.00	-3.1	-3.0	-3.5	pb26	+0.33	+1.71	0.00	-1.5	-0.5	-1.8
pb11	+1.37	+3.27	-0.60	-2.7	-1.9	-2.8	pb27	-0.48	+1.18	-2.39	-2.9	-3.0	-3.8
pb12	+1.63	+3.20	+1.63	-2.3	-2.0	-2.4	pb28	-1.31	0.00	-1.70	-3.5	-1.5	-3.6
pb13	+0.39	+1.29	-0.91	-4.4	-4.2	-5.1	pb29	0.00	0.00	0.00	0.0	0.0	0.0
pb14	-0.95	+0.81	-2.31	-4.0	-4.2	-4.6	pb30	0.00	0.00	0.00	0.0	0.0	0.0
pb15	-0.41	+0.96	-2.77	-3.5	-3.3	-4.4	pb31	0.00	0.00	-0.30	+1.0	+0.4	0.0
pb16	+1.08	+2.13	-0.47	-3.9	-4.4	-5.0	pb32	+1.43	+2.43	-0.62	-1.5	-1.0	-1.4
Avg.	+1.84	+2.89	+0.49	-2.4	-2.2	-2.7	*Avg.*	+0.52	+1.24	-0.83	-1.9	-1.5	-2.2

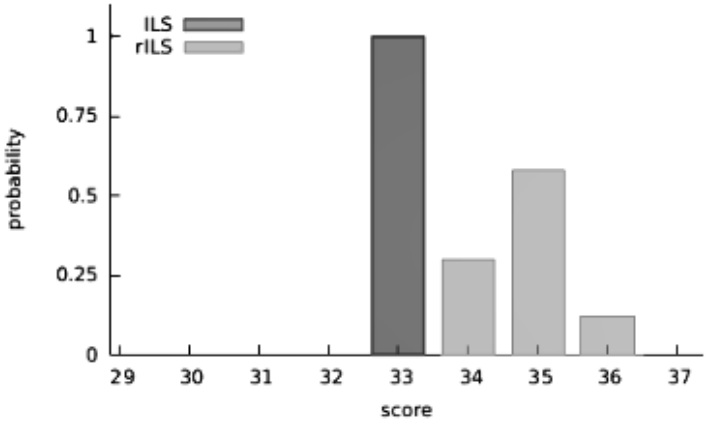


Figure 6.1: Score probability distribution (100 independent runs) typical to Table 6.2 problem instances where rILS's median score is higher than ILS's score.

perturbation made the search diverse and, thus, more likely to visit different local optima or take a different trajectory when a solution is revisited (i.e., the search is made robust to cycling). When supposed to be a guided random-walk in the space of local optima, we rather found ILS similar or worse than random-walks (MRLS and rILS). Consequently, ILS's guidance components need to be tuned to some minimum level.

6.6 Conclusion

In this chapter, we showed how to experimentally evaluate ILS so to tell if and where it needs tuning. Our contribution was, first, to show how to derive two trivial baselines: multi-restart local search (MRLS) and randomized ILS (rILS), from the same ILS we want to evaluate. Second, we implemented and evaluated an ILS, that is originally competitive on the TOPTW, on a similar NP-hard combinatorial optimization problem (our MRTA problem). The two proposed metaheuristic baselines are relevant to evaluate ILS's search guidance components, namely the perturbation and the acceptance criterion together. The

evaluation we suggest is easy to carry out, because there are little implementation and configuration effort. Furthermore, it proved to be utile as to check if ILS is minimally tuned. Consequently, the designer is spared the free parameter tuning burden, when at the first place, it is ILS's search guidance components that need tuning. We will build upon ILS based on our findings here to better approach our problem, MRTA-STC, in the next chapter.

Chapter 7

Enhanced Iterated Local Search for MRTA-STC

> It's easy to implement such
> quick-and-dirty SLS algorithms, but not
> the state-of-the-art ones
>
> Mauro Birattari

In this last chapter, we approach MRTA-STC with a metaheuristic of the type iterated local search (ILS). We propose an enhanced ILS, which we build on top of Vansteenwegen et al.'s ILS [61], after we have adapted the implementation to our problem, guided by the preliminary experimental study [67] (Chapter 6). Enhanced ILS empirically proves to be better than the baseline ILS [61], on MRTA-STC, with regard to solution quality and response time. Furthermore, enhanced ILS is competitive, with the baseline ILS, when the search is stopped at different time points. Moreover, enhanced ILS, which is of a stochastic nature, is robust to run replication and only requires a little tuning effort to outperform the baseline ILS. The chapter is organized as follows. In Section 7.1, we introduce our last approach to the problem. In Section 7.2, we detail the construction module of our ILS. In Section 7.3, we present the new perturbation module. In Section 7.4, we give enhanced ILS's algorithm, discuss how it works and why we think it is an improvement over the baseline ILS. In Section 7.5, we experimentally evaluate the proposed metaheuristic against the baseline ILS.

7.1 Introduction

While greedy and insertion-based heuristics (Chapter 5) perform better than the market-based approach (Chapter 5); the former (heuristics that construct tours) cannot escape local optima. Metaheuristics, such as iterated local search (Chapter 6), come to find better solutions, by embedding a local search (or a heuristic), restarting the search and using other search tricks. To better tackle MRTA-STC difficulty (particularly when a time critical response is required), we propose an anytime approach that quickly returns a suboptimal solution, yet it can find a better solution if ran longer. Namely, we devise an enhanced ILS [68] that builds upon ILS [61], based on the preliminary study we did on ILS (the baseline)

when adapted for our problem [67] (Chapter 6). The metaheuristic runs repeatedly in two phases: a construction, that generates local optima, and a perturbation, that tries to escape them to hopefully find a global optimum or at least a local optimum of a better quality. Adopting the routing and scheduling terminology, we recall that we designate an agent's task assignment by a visit and an agent's set of task assignments by a tour.

7.2 Construction Step: an Insertion-based Heuristic

This phase builds a candidate solution, from some partial solution, by a heuristic that inserts visits into all tours simultaneously. Let s_i, f_i and T_i respectively denote the start time, the finish time, and the service time of visit i. We denote – with some abuse of notation – by $\rho(h,i)$, the travel time between locations h and i and by $\rho(-1,h)$ the travel time between the agent initial location and l_h, the location of some visit that is the first in the tour. The insertion of visit i between h and j delays visit j and the visits that follow j, in the tour, by shift_i:

$$\text{shift}_i = \rho(h,i) + T_i + \rho(i,j) - \rho(h,j). \tag{7.1}$$

Given that the tasks have deadlines, before delaying a visit, we ensure that it remains feasible after the visit under consideration is inserted. For that, we compute maxshift_j, the maximum delay that is allowed on visit j,

$$\text{maxshift}_j = \max\left\{0, \left\{\begin{array}{ll} d_j - (s_j + T_j) & \text{if } j \text{ is the last} \\ \min\{d_j - (s_j + T_j),\ \text{maxshift}_{j+1}\} & \text{otherwise} \end{array}\right\}\right\} \tag{7.2}$$

where d_j is the deadline of visit j – with some abuse of notation – and maxshift_{j+1} is the maximum affordable delay on the visit that follows j. A visit can be delayed as late as its task's deadline permits, when it is the last in the tour. Otherwise, the visit can be delayed no later then its task's deadline and the maximum shift of the visit that follows (Eq. 7.2).

The insertion of visit i between h and j is feasible if, and only if, i is feasible (Eq. 3.1, Chapter 3) and the resulting delay to j, consequently, the delays to the visits that follow j, do not exceed the maximum allowable delay:

$$\text{shift}_i \leq \text{maxshift}_j \tag{7.3}$$

A visit maybe feasible for insertion in many tour positions. The desirability of inserting visit i, noted as δ_i, is inversely proportional to the delay that results from the insertion (Eq. 7.4). Let I denote the set of alternative visit insertions that are possible for the currently unallocated tasks ($K_{pending}$). Then, let I_k denote the set of feasible visits to some task $k \in K_{pending}$. Finally, I^* denotes the set of the most desirable insertions to pending tasks ($I^* \subseteq I$); which is obtained by calculating i_k^*, the visit to task k that is the most desirable with regard to all the tours (Eq. 7.5), for every pending task k. The visit to select for insertion, noted i^*, has the lowest delay among the most desirable visits to the currently

pending tasks (Eq. 7.6).

$$\delta_i = 1/\text{shift}_i \tag{7.4}$$

$$i_k^* = \underset{i \in I_k}{argmax}\ \delta_i \tag{7.5}$$

$$i^* = \underset{i \in I^*}{argmax}\ \delta_i \tag{7.6}$$

After inserting a visit, we update the maximum shift for all the visits in the concerned tour. As to the visits that come after the insertion, we further update their start times and finish times, by adding the insertion delay (Eq. 7.7,7.8).

$$s_j = s_j + \text{shift}_i \tag{7.7}$$

$$f_j = f_j + \text{shift}_i \tag{7.8}$$

Algorithm 14: Construction step in enhanced ILS

Input: partial solution S, set of tasks $K_{pending}$
Output: new solution S'

```
1  S' ← S
2  while K_pending ≠ ∅ do
3  |  forall k ∈ K_pending do
4  |  |  Compute all possible visits to task k (Eq. 3.1, 7.3)
5  |  |  Find the best visit to task k (Eq. 7.5)
6  |  Find the best visit to insert, i* (Eq. 7.6)
7  |  if i* = null then return S'
8  |  else Insert i* into S' and
9  |  update K_pending
10 |  Compute the start time s_i* and the finish time f_i*
11 |  forall the visits j after i* do
12 |  |  Update s_j, f_j and maxshift_j (Eq. 7.7, 7.8, 7.2)
13 |  Compute maxshift_i*
14 |  forall the visits h before i* do
15 |  |  Update maxshift_h
```

The construction step starts from a given solution and iterates until all tasks are visited, or no further insertion is possible (Algorithm 14). Each time, it adds, to the current solution, the best visit among the possible visits to the currently pending tasks. Then, this phase updates the set of pending tasks and the tour where the visit insertion took place, by re-computing the visits' start times, the visits' finish times, and the visits' maximum shifts. The maximum shifts are updated in the inverse order of visits (Eq. 7.2).

7.3 Perturbation Step: an alteration of the current solution

The perturbation phase modifies the current solution so that the construction phase can find another solution. The perturbation strength and the perturbation start define the modification to make. The adjustment of these perturbation parameters affects the search tendency (exploration or exploitation). To perturb the solution, we remove from each tour

r sequential visits, in a circular way, starting from some visit at index x. Namely, we remove visits from index x to $x + r - 1$. When we reach the end of the tour, we continue from the beginning (i.e., from index 0 to $y = (x + r)\ \%\ tl - 1$, where $\%$ is the remainder of division operator and tl is the length of the tour). The perturbation strength is then nr. Removing visits requires a backward shift, $\text{backshift}_{x,r}$, of the visits that follow the removal, unless the visits are removed from the end of the tour (7.9).

$$\text{backshift}_{x,r} = \begin{cases} 0 & \text{if } x + r = tl \\ \sum_{i=0}^{y} [\rho(i-1,i) + T_i] + \rho(y, y+1) - \rho(-1, y+1) & \text{if } x + r > tl \\ \sum_{i=x}^{x+r-1} [T_i + \rho(i, i+1)] + \rho(x-1, x) - \rho(x-1, x+r) & \text{otherwise} \end{cases} \tag{7.9}$$

When visits are removed from the beginning or the middle of the tour, the visits that follow are shifted toward the beginning. The start and finish times of the shifted visits are updated as follows (where $i \in \{x + r \dots tl - 1\}$ when $x + r < tl$ and $i \in \{y + 1 \dots x - 1\}$ when $x + r > tl$),

$$s_i = s_i - \text{backshift}_{x,r} \tag{7.10}$$

$$f_i = f_i - \text{backshift}_{x,r} \tag{7.11}$$

The removal of visits requires to update the maximum shifts of the visits that remain. When visits are removed from the middle of the tour (i.e., when $x + r < tl$), the maximum shifts of the visits i that follow the removal ($i \in \{x + r, \dots, tl - 1\}$) are computed as follow:

$$\text{maxshift}_i = \text{maxshift}_i + \text{backshift}_{x,r} \tag{7.12}$$

Our perturbation phase removes from every tour a number of visits that is propor-

Algorithm 15: Perturbation step in enhanced ILS

Input: current solution S, free parameter *perturbationStrengthCoef*
Output: partial solution S'

```
r ← (score(S) × perturbationStrengthCoef)/n
S' ← S
forall tour ∈ S' do
    x ← random number ∈ [0, length(tour))
    Remove the r first visits starting at index x in tour
    Update K_pending
    Compute backshift_{x,r} in tour (Eq. 7.9)
    forall the visits j ≥ x + r do
        Update s_j, f_j and maxshift_j (Eq. 7.10–7.12)
    forall the visits i < x do
        Update maxshift_i (Eq. 7.2)
```

tional to the number of inserted tasks ($score(S)$), the perturbation strength coefficient (perturbationStrengthCoef) and the number of agents (n) (Algorithm 15). Likewise, the perturbation strength is somehow evenly divided among the tours, it adapts to the search

progression and to the problem instance's difficulty (number of agents, task/agent locations, deadlines/capacities setting). Namely, as the number of visits increases or decreases, the perturbation strength changes accordingly. Each time and for each tour, the perturbation starts from an arbitrary index (x) (Algorithm 15). By doing so, the perturbation is very likely to output a different solution S', if it happens to have again solution S as an input. Consequently, our ILS should avoid cycles and explore the search space better then ILS [61].

7.4 The metaheuristic: Enhanced Iterated Local Search (enhILS)

Algorithm 16: Enhanced iterated local search (enhanced ILS)

Input: $maxIter$, $perturbationStrengthCoef$
Output: best-found solution S^*

```
1 S' ← ∅
2 S* ← S'
3 iter ← 1
4 while iter ≤ maxIter do
5 |   S ←Construction(S')    (Algorithm 14)
6 |   if score(S) > score(S*) then S* ← S
7 |   S' ←Perturbation(S)    (Algorithm 15)
```

Enhanced ILS (Algorithm 16) is configured with two parameters: (1) the maximum number of iterations to do ($maxIter$) which, contrary to $maxTrials$ of ILS [61], enables the control of the runtime; (2) a coefficient that controls the perturbation strength at the tour level (perturbationStrengthCoef). Enhanced ILS iterates until all tasks are visited or the maximum number of iterations is reached. Each time, it constructs a new solution, S, from a partial solution, S' (initially empty), that is based on the current solution, S. The best solution found, S^*, is updated with the new solution S if necessary. Then, S is altered so that the next iteration may escape the local optimum towards a possibly better one.

If we find better solutions, that could only be a consequence of a better exploitation and/or exploration of the search space. Since we kept ILS's construction heuristic (exploitation component) and the same acceptance criterion, it would clear that it is the perturbation procedure of enhanced ILS that would made the difference. We think that the diversity we added to the perturbation phase should make enhanced ILS avoid cycles during the search (likely to happen in deterministic ILS). Furthermore, by relying on small or moderate perturbations, we should avoid searches like random restarts (MRLS), while progressively doing the exploration. Finally, the randomly sampling the neighbor solutions should improve the search exploitation.

7.5 Experimental evaluation

We experimentally evaluate enhanced ILS on synthetic MRTA-STC instances. We generated the problem instances based on TOPTW instances; specifically, according to benchmarks that are based on Solomon's VRPTW instance (Chapter 6). Optimal solutions to the MRTA-STC instances are not available. An alternative way to evaluate our approach is

to compare it with the state-of-the-art TOPTW's approaches. Unfortunately, such a comparison is not straightforward: we need to correctly code each competitor's metaheuristic, fairly tune them, and conduct experiments with multiple run replications on a large set (96 instances) [1]. Instead, we select, as a baseline, TOPTW's approach that we find appropriate to MRTA-STC, yet which is competitive.

The comparison of the state-of-the-art approaches to TOPTW was conducted by Hu et al. [63] and recently by Gunawan et al. [69]. The authors adapted the computation times of the competitors methods according to their CPU speeds. The comparisons was carried out on two TOPTW benchmarks with up to four tours. We only consider the benchmark that is based on Solomon's instances, as is our testset. We compare the performance averaged per group of instances similarly to [63, 69]. The iterative three-component heuristic [63] tends to get the lowest score gap to the best-known solutions; however it has a high computation time. The hybridization of Simulated Annealing with ILS [69] is comparable to the iterative three-component heuristic, but only for long computation times. ILS [61] is appropriate for real time, since it can in the order of seconds return solutions with a low score gap to the best-known—the average score gap was only 1.76% and 2.34%, respectively, for the second group and the first of Solomon's instances [2], while the average response time is tens to hundreds times faster than the ant colony system [70] and the iterative three-component heuristic. The hybrid metaheuristic of Labadi et al. [71] is a few times slower than ILS, but has a slightly better average score gap (about 1.2% better on average in each instances group [63]). The metaheuristics of Labadi et al. [71] and Vansteenwegen et al. [61] are suitable baselines for time critical applications, since both have prompt response times and low score gaps compared to the best-known. Nonetheless, ILS [61], has the best response time. Thus, we select it as the baseline (as implemented in [67]) in the evaluation of enhanced ILS.

7.5.1 Experimental Instances and Setup

We keep the testset described in Chapter 6. We coded enhanced ILS and the baseline ILS in Java. We ran the experiments on a laptop equipped with an Intel(R) Core(TM) i7 CPU clocked at 2.20 GHz, 8 GB of volatile memory and running under Linux Ubuntu 14.04. We replicated enhanced ILS run on a problem instance 10 times.

7.5.2 Parameter Tuning

We manually tuned the algorithms in a trial-and-error fashion. We stopped the tuning when the quality of the solution did no longer improve without significantly increasing the computation cost. We found the best parameters setting for ILS ($maxTrials$, maxPerturbationStrengthCoef) to be $(600, 1/2)$ and $(600, 2/5)$, respectively, for group 1's and group 2's instances. Based on the run replication with the median score, we found the best parameters setting for enhanced ILS ($maxIter$, perturbationStrengthCoef)

[1] Replicating others' work is error-prone. Often, the algorithms are not described clearly enough to be replicated, or their tuning cannot be reproduced. For instance, some authors tune their algorithms manually without detailing how well the tuning was performed; other authors diverge in the experience and effort put into tuning.

[2] Gunawan et al. [69], though more recently, report ILS score gaps that are even lower than Hu et al. [63]

to be $(3000, 1/3)$ and $(5000, 1/8)$, respectively, for group 1's and group 2's instances. We found marginal improvements beyond 15 seconds of execution.

7.5.3 Results

Table 7.1 summarizes the results of the experimental comparison of enhanced ILS and ILS. The score gap is reported in percentage (%) and the computation time gap in milliseconds (ms), per problem (Eq. 7.13).

$$\begin{aligned} ScoreGap = \quad & (score_{enhILS} - score_{ILS}) \div score_{ILS} \times 100\% \\ TimeGap = \quad & \begin{cases} time_{ILS} - time_{enhILS} & \text{if } ScoreGap = 0 \\ (runtime_{runner\text{-}up} - time_{winner}) & \text{otherwise,} \end{cases} \end{aligned} \tag{7.13}$$

In Eq. 7.13, $score_{enhILS}$ and $score_{ILS}$ are the solution qualities of the proposed and the baseline metaheuristics, respectively. The computation time that is actually necessary to hit the final solution is noted $time_{ILS}$ and $time_{enhILS}$. As to $runtime_{runner\text{-}up}$, it is the runtime of the metaheuristic that underperformed, while $time_{winner}$ is the computation time that was actually necessary to the metaheuristic that performed the best (on the problem at hand) to find its solution. Then the time gap, $TimeGap$, measures how much extra time the runner-up approach was given to catch up the other approach, if any (Eq. 7.13). In so doing, we can, on one hand, check if the runner-up metaheuristic was given a fair computation time. On the other hand, we can measure how prompt the winner approach hit its final solution, relatively to the runner-up. In Table 7.1, column one lists the problems, columns two to four report the score gaps respectively obtained from the median, best and worst run replications of enhanced ILS. Columns five through seven, respectively, report the time gap for enhanced ILS run replications with the median, best and worst score gaps. Each line reports the average result over the instances of the corresponding problem configuration (Table 7.1). Score gaps are positive when our approach performs better than the baseline. When the time gaps are positive, as well, our approach managed to find solutions of better quality earlier (Eq. 7.13).

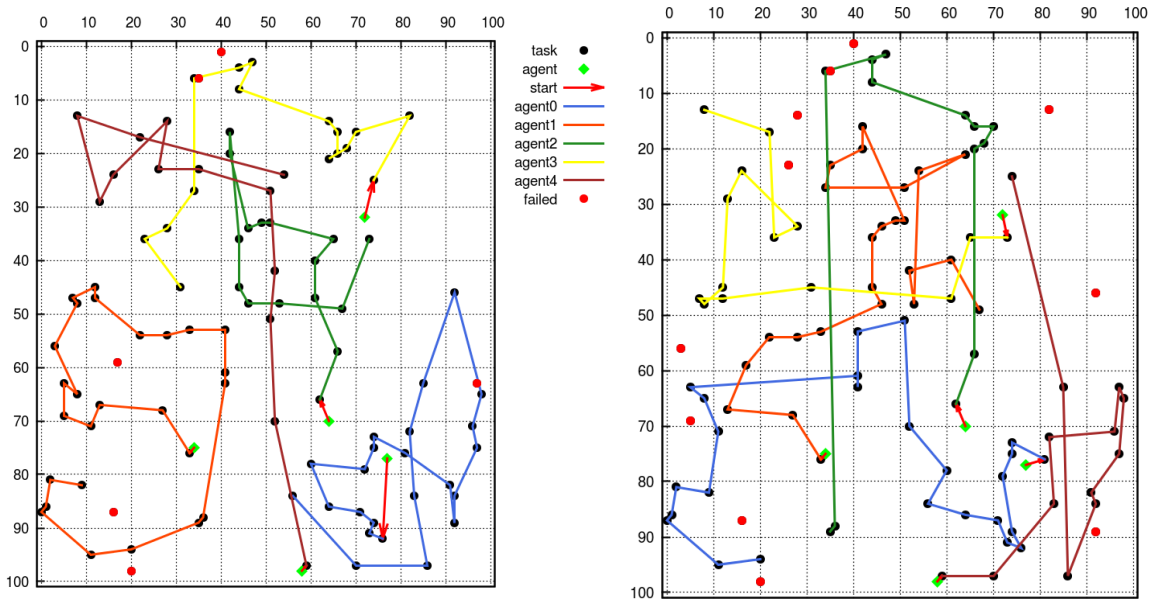
Compared to ILS, enhanced ILS generally finds solutions of better quality, within shorter times (Table 7.1). Even in the worst run replication, enhanced ILS is generally better (it is at least as effective as ILS on 13 out of the 16 problems, in each group). In the median case, enhanced ILS is on average 2.24% and up to 7.21% more effective, in allocating tasks. When we consider all the run replications, enhanced ILS almost always improves the solution quality (it did not only on two problems in group 2). The total average score gap is slightly improved in the best run replication compared to the median run. Nonetheless, the score gap is further improved on half of the problems, in both groups (Table 7.1). Enhanced ILS tends to have a response time that is shorter than ILS (Table 7.1). In the median run replication, ILS is given, on average, one second and up to 5.5 seconds to catch up with enhanced ILS.

Undoubtedly, enhanced ILS did better (than baseline ILS) thanks to the perturbation procedure. It turns out that adapting the perturbation strength (to the incumbent

Table 7.1: Performance of enhanced ILS *vs.* baseline ILS, averaged per problem.

Problem	Score gap (%)			Time gap (ms)		
	median	best	worst	median	best	worst
r11a2	0.0	0.0	0.0	9	15	15
r12a2	1.38	2.76	0.0	1558	1464	4
r13a2	1.43	1.43	1.43	1174	1366	1364
r14a2	5.18	5.18	3.63	1080	1141	1202
r11a3	3.1	3.1	3.1	2760	2783	2717
r12a3	4.23	4.23	3.26	2480	2418	2497
r13a3	5.21	6.51	1.95	1418	1607	2368
r14a3	3.8	3.8	1.52	2133	2122	2083
r11a5	2.68	3.25	1.34	4688	3532	4917
r12a5	0.0	1.9	-0.76	-313	3883	5389
r13a5	2.56	2.56	0.0	3358	4143	259
r14a5	3.31	4.02	1.65	2097	2968	3969
r11a7	1.3	1.3	-0.52	3443	3878	11234
r12a7	0.0	0.95	-0.41	-207	5221	8287
r13a7	0.97	1.94	0.55	4973	3401	2927
r14a7	2.64	2.64	1.09	5547	7041	6591
average	2.36	2.85	1.11	2262	2936	2378
minimum	0.0	0.0	0.0	-313	15	4
maximum	5.21	6.51	3.63	5547	7041	6591

Problem	Score gap (%)			Time gap (ms)		
	median	best	worst	median	best	worst
r21a2	1.56	1.56	-0.67	1303	1901	3815
r22a2	4.02	4.73	1.65	1290	1539	2508
r23a2	1.74	1.74	0.99	1428	1315	2232
r24a2	7.21	7.21	5.11	1143	1024	1330
r21a3	1.11	1.58	0.0	-215	1678	125
r22a3	2.59	3.73	0.97	14	-657	1892
r23a3	1.72	2.4	0.69	2183	1751	2890
r24a3	0.8	1.99	-1.19	2526	3236	2893
r21a5	2.4	2.4	1.67	-102	-699	1294
r22a5	2.84	3.6	2.18	879	-975	552
r23a5	3.11	4.3	2.39	1930	2118	2996
r24a5	2.23	3.15	0.52	1544	1437	2972
r21a7	0.0	0.0	0.0	15	15	15
r22a7	0.0	0.0	0.0	17	10	18
r23a7	0.0	0.0	-0.3	-12	-476	2417
r24a7	2.49	3.12	2.08	487	1013	1577
average	2.11	2.59	1.01	902	889	1569
minimum	0.0	0.0	-0.67	-215	-975	15
maximum	7.21	7.21	5.11	2526	3236	2996



(a) enhILS (b) ILS

Figure 7.1: EnhILS *vs.* ILS solution map on a MRTA-STC problem instance with five agents, 100 tasks and slack deadlines.

solution and to the problem instance), parameterizing the perturbation and injecting some randomness into the perturbation (the case of enhanced ILS), is better than doing cyclic incremental deterministic perturbations which are upper bounded by a free parameter (the case of baseline ILS). We think that our perturbation procedure made the search unlikely to recycle. Interestingly, enhanced ILS's perturbation proves particularly helpful on MRTA-STC problem instances with (relatively) short tours (group 1). Indeed, enhanced ILS achieves at least 2.5% score improvement in nine group 1's problems, compared to five group 2's problems (Table 7.1).

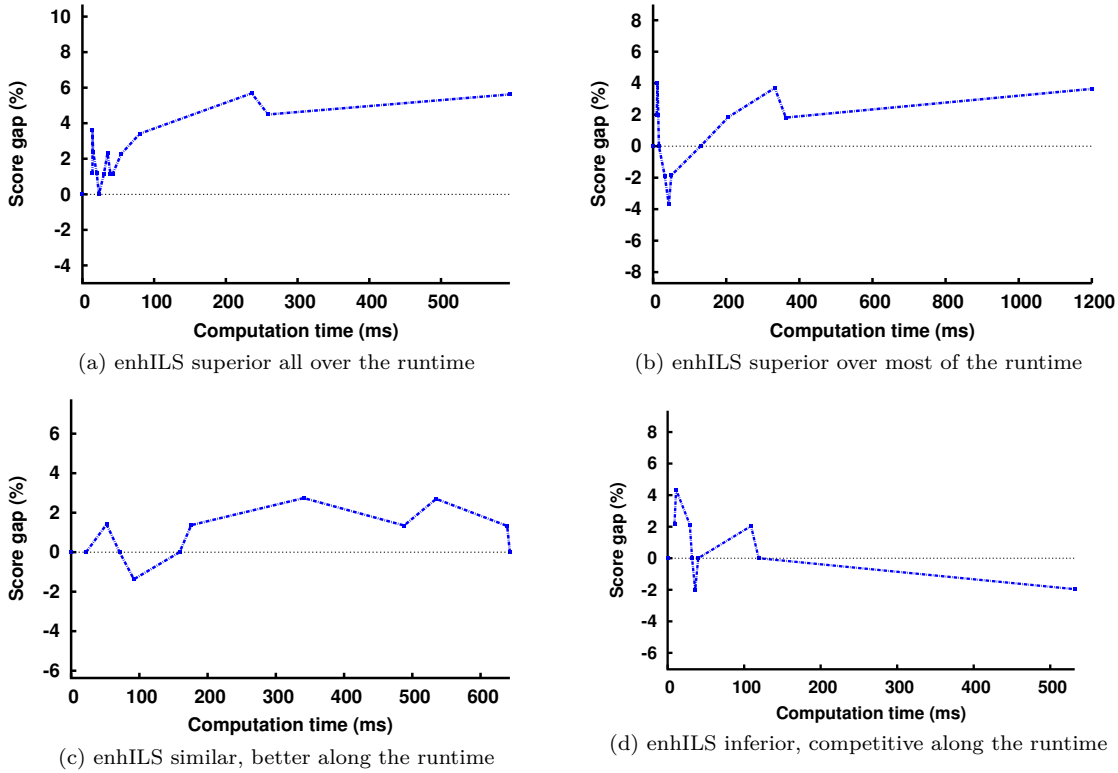
Table 7.2 details the median performance results of enhanced ILS. The columns from left to right, respectively, lists the problem instances names, the solution quality (score), the score gap (enhanced ILS vs. ILS), enhanced ILS's computation time and runtime. Strictly positive score gap values, which account for the superiority of our metaheuristic, are noted in bold. Enhanced ILS finds solutions of better quality than ILS's (about) 68% of the time (on 65 out of the 96 instances). Enhanced ILS fell shorter than the baseline only on three instances: r12a500, r12a702 and r13a700 (Table 7.2). The score improvement is up to about 10%. On average, enhanced ILS performs slightly better on instances with tight deadlines (group 1). Both ILS find the optimal solution of at least 10 group 2's problem instances (when 100 tasks are allocated, Table 7.2). Therefore, we cannot draw a conclusion on such (seemingly) easier to solve problem instances. Enhanced ILS is ran, on average, less than five seconds and no more than 15 seconds. The response time is less then that (Table 7.2).

Figure 7.1 illustrate the performance of enhanced ILS vs. ILS through a sample of solutions maps. These depicts two solutions obtained by running enhanced ILS and ILS on an MRTA-STC problem instance with 100 tasks, five agents and slack task deadlines. Unallocated tasks are marked in red (Figure 7.1). We can see that enhILS completed five more tasks than ILS— enhILS's agent number 0 failed to complete two more tasks than ILS's agent 0 did, while agents number $1, 2, 3$ and 4, respectively, complete two, three, two

Table 7.2: Detailed performance of enhanced ILS (median run) versus ILS.

Instance	Score	Gap (%)	Time (ms)	
			Comput.	Run
r11a200	23.0	0.0	42	1732
r11a201	23.0	0.0	13	1581
r11a202	23.0	0.0	2	1832
r11a300	31.0	**3.33**	119	2481
r11a301	38.0	**2.7**	185	4243
r11a302	31.0	**3.33**	4	2594
r11a500	52.0	**1.96**	1210	5962
r11a501	52.0	**1.96**	701	6297
r11a502	57.0	**3.64**	1200	9261
r11a700	78.0	**2.63**	8069	13987
r11a701	79.0	**1.28**	3952	14358
r11a702	76.0	0.0	643	10956
r12a200	22.0	**4.76**	10	1114
r12a201	21.0	0.0	525	1077
r12a202	23.0	0.0	9	1954
r12a300	30.0	**3.45**	22	2129
r12a301	36.0	**2.86**	16	3790
r12a302	30.0	**7.14**	1513	2177
r12a500	50.0	-1.96	532	5683
r12a501	52.0	**1.96**	764	5244
r12a502	56.0	**1.82**	3270	8873
r12a700	73.0	**2.82**	4845	9382
r12a701	76.0	0.0	1090	9440
r12a702	73.0	-1.35	2809	9412
r13a200	22.0	0.0	2	934
r13a201	22.0	**4.76**	7	1063
r13a202	20.0	0.0	261	851
r13a300	31.0	**3.33**	76	1815
r13a301	35.0	**6.06**	240	2836
r13a302	31.0	**6.9**	430	1871
r13a500	50.0	**4.17**	464	4621
r13a501	52.0	**4.0**	1397	5472
r13a502	54.0	**1.89**	1056	5433
r13a700	73.0	-1.35	1759	8799
r13a701	75.0	**2.74**	7782	8854
r13a702	71.0	**1.43**	6043	8527
r14a200	21.0	**10.53**	397	864
r14a201	20.0	0.0	4	755
r14a202	20.0	**5.26**	10	1064
r14a300	25.0	**8.7**	32	1998
r14a301	31.0	**3.33**	73	3036
r14a302	26.0	0.0	22	1928
r14a500	42.0	**2.44**	85	3879
r14a501	43.0	**2.38**	2069	5034
r14a502	46.0	**4.55**	698	5761
r14a700	67.0	**3.08**	829	8711
r14a701	64.0	**1.59**	3477	9961
r14a702	67.0	**3.08**	5681	9556
average		2.52	1342	4983
maximum		10.53	8069	14358
minimum		-1.96	2	755

Instance	Score	Gap (%)	Time (ms)	
			Comput.	Run
r21a200	46.0	0.0	2764	5019
r21a201	46.0	**2.22**	9	4117
r21a202	45.0	**2.27**	927	3854
r21a300	59.0	0.0	170	7068
r21a301	72.0	**1.41**	1762	7638
r21a302	61.0	**1.67**	103	6865
r21a500	97.0	**2.11**	918	7448
r21a501	97.0	**4.3**	1993	8056
r21a502	100.0	**1.01**	214	441
r21a700	100.0	0.0	14	15
r21a701	100.0	0.0	15	22
r21a702	100.0	0.0	15	17
r22a200	44.0	**7.32**	615	3172
r22a201	43.0	0.0	2026	3298
r22a202	45.0	**4.65**	119	3392
r22a300	60.0	**5.26**	281	5428
r22a301	71.0	**2.9**	1558	6769
r22a302	59.0	0.0	551	5695
r22a500	93.0	**4.49**	596	6889
r22a501	94.0	**3.3**	634	7439
r22a502	96.0	**1.05**	1801	7904
r22a700	100.0	0.0	15	16
r22a701	100.0	0.0	40	20
r22a702	100.0	0.0	15	17
r23a200	41.0	0.0	1614	2767
r23a201	41.0	**2.5**	642	2774
r23a202	41.0	**2.5**	770	2738
r23a300	55.0	**1.85**	1708	3859
r23a301	67.0	**1.52**	2417	5254
r23a302	56.0	**1.82**	1019	3862
r23a500	81.0	**3.85**	3570	6004
r23a501	89.0	**4.71**	1082	6227
r23a502	89.0	**1.14**	2973	5886
r23a700	100.0	0.0	23	24
r23a701	100.0	0.0	1757	61
r23a702	100.0	0.0	15	17
r24a200	36.0	**9.09**	149	1702
r24a201	35.0	**2.94**	123	1609
r24a202	36.0	**9.09**	233	1754
r24a300	46.0	0.0	2111	3036
r24a301	56.0	0.0	3255	3982
r24a302	50.0	**2.04**	7	3309
r24a500	76.0	0.0	1453	6973
r24a501	77.0	**4.05**	1478	6595
r24a502	81.0	**2.53**	1317	6322
r24a700	99.0	**2.06**	234	8220
r24a701	98.0	**3.16**	631	7608
r24a702	99.0	**2.06**	2151	8337
average		2.1	998	4157
maximum		9.09	3570	8337
minimum		0.0	7	15



(a) enhILS superior all over the runtime

(b) enhILS superior over most of the runtime

(c) enhILS similar, better along the runtime

(d) enhILS inferior, competitive along the runtime

Figure 7.2: Patterns of score gap evolution over time (enhanced ILS *vs.* ILS)

and zero more tasks—. Furthermore, enhanced ILS's tours tend to have the allocated tasks clustered (see tours 0, 1 and 2, Figure 7.1(a)), while ILS's tours tend to have longer travel distances, agent routes that intersect and less tasks allocated, relatively to enhILS's tours (e.g., compare tour 2 and tour 3 of enhanced ILS with their peers in ILS, Figure 7.1). We think enhILS explored the solution space better than ILS, so to be able to, not only allocate more tasks, but also (sometimes) find solutions that are better visually coherent (Figure 7.1).

To further evaluate our approach, we plot the score gap evolution over time. For that, we consider enhanced ILS's median run and ILS (single) run on the 96 problem instances. Figure 7.2 shows four representative patterns of score gap evolution over time—we omit a fifth pattern, with 26% of occurrences, since therein the score gap is null and does not evolve over time. Enhanced ILS is "often" superior to ILS all over the runtime (Figure 7.2-(a)). Specifically, in 56.3% of the instances, not only does enhanced ILS return a better solution, but it can also do that any time. Sometimes (11.5% cases), enhanced ILS achieves a score that is better than ILS's by the end of the runtime, while being, generally, superior during the runtime (Figure 7.2-(b)). Enhanced ILS performs equally to ILS by the end of the run in 29% cases (Figure 7.2-(c)). Interestingly, when enhanced ILS underachieves by the end (3% of the cases), it is still competitive with ILS along the runtime (Figure 7.2-(d)). We can fairly conclude that enhanced ILS is very likely to achieve a solution quality that is better than ILS's, even when one interrupts the run at different cutoff times.

We do a statistical study on the performance of our approach. We consider 30 observations per problem configuration, which we obtain by replicating enhanced ILS run on each problem configuration's instances. Then, we compute the probability that enhanced ILS improves, not improve, or deteriorates, respectively, the solution quality achieved by

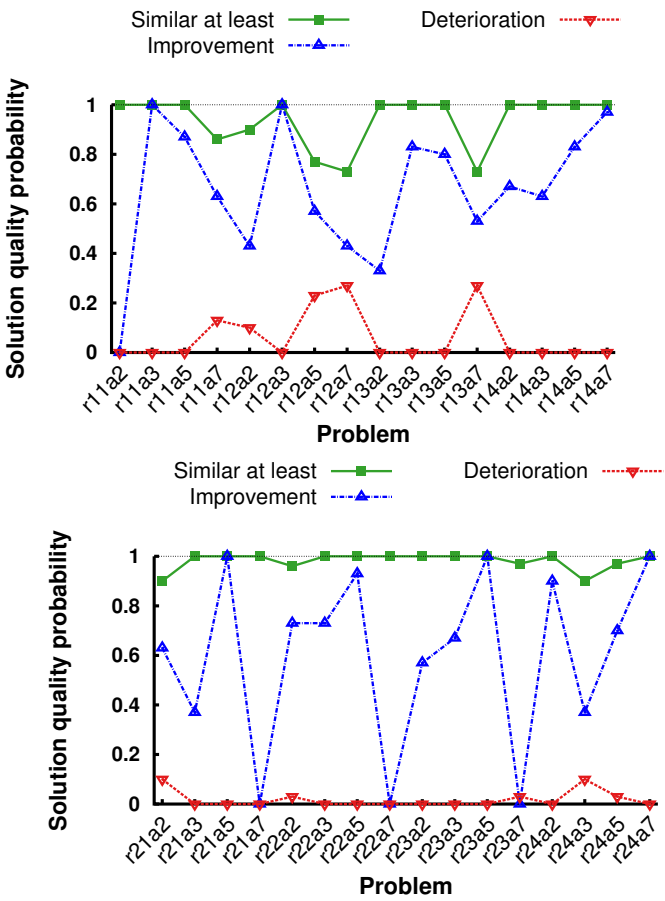


Figure 7.3: Probability that enhanced ILS improves ILS's solution quality.

ILS (Figure 7.3). Enhanced ILS is at least 50% likely to improve ILS's solution in most problem instances for each group. Enhanced ILS is overall 95% likely to perform at least as well as the baseline. Practically, enhanced ILS never scores below ILS, particularly for group 2's instances (Figure 7.3). The probability that enhanced ILS deteriorates the solution quality (achieved by ILS) is low or null, on the majority of the problems (Figures 7.3). Although our metaheuristic sometimes performs similarly to, or worse than, the baseline; it is interesting that it can find a better solution – almost all of the time – when we can afford many run replications—in contrast, replicating the baseline run does not help to improve the solution quality. On average, enhanced ILS is 66% likely to improve ILS's solution quality and only 6% of chance to deteriorate it on group 1's instances. On group 2's instances, enhanced ILS is 57% likely to improve solution quality and only 4% likely to deteriorate it. We conclude that, though stochastic, enhanced ILS is practically better than ILS (with regard to solution quality) when it is ran once. Enhanced ILS is further better if we do multiple runs.

7.5.4 Sensitivity to Parameters Setting

A metaheuristic's performance partly depends on how its free parameters are set. To evaluate the impact of the free parameters setting on our approach's performance, we measure the sensitivity of the score gap (enhanced ILS vs. ILS) to the free parameters setting (Figure 7.4). The free parameters settings that we consider to configure enhanced

ILS (noted as config, Figure 7.4) are given by Table 7.3. We configure ILS by hand the best we could (Section 7.5.2). Enhanced ILS generally performs better than ILS (with regard to solution quality), independently of how its free parameters are set. Specifically, enhanced ILS performs worse than ILS only on four problems and with a few free parameters settings in each group (Figure 7.4). On average, the parameters tuning results in a score gain of 2.23%, only. Therefore, enhanced ILS's performance is barely affected by the free parameters setting. That is, our approach's performance is, generally, superior to the baseline's best independently of parameters tuning.

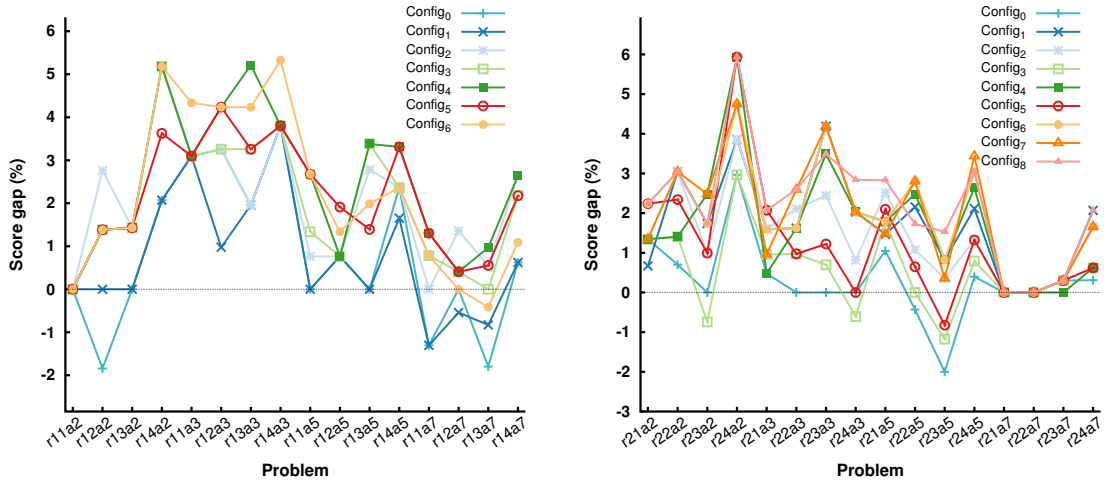


Figure 7.4: Enhanced ILS vs. tuned ILS (Section 7.5.2) score gap sensitivity to the free parameters setting (config), when we consider enhanced ILS median score

Table 7.3: Some free parameters settings we have tried during enhanced ILS tuning

Config.	group1's instances		group2's instances	
	maxIter	maxPerturbation StrengthCoef	maxIter	maxPerturbation StrengthCoef
0	6000	0.1	1000	0.5
1	5000	0.125	3000	0.125
2	4000	0.2	2000	0.25
3	3000	0.25	1500	0.4
4	3000	0.33	4500	0.1
5	2100	0.4	1700	0.33
6	1500	0.5	4000	0.125
7	-	-	5000	0.125
8	-	-	4000	0.2

7.6 Conclusion

In this chapter, we showed how a simple metaheuristic, iterated local search, can be applied to our MRTA problem, to effectively tackle its complexity in real time. We showed how the approach works in detail, so it can easily be reproduced. We discussed how our ILS relates and differs from a competitive ILS for TOPTW problem, while we showed where we think the improvement comes from. We experimentally evaluated our approach on synthetic test instances, which we have generated. We found that enhanced ILS, our approach, produces

solutions to MRTA-STC that have a better quality then the baseline's, while showing a prompter response time. Our approach is barely sensitive to parameters tuning. It is fairly robust to run replication as well. The approach we have proposed still performs well when it is interrupted at different cutoff times. Future work directions should investigate the use of a sophisticated acceptance criterion, multiple construction heuristics and/or an adaptive parameter selection (for tuning). We think we should as well gather realistic test data, or at least evaluate the proposed ILS on other elaborated benchmarks. Though our ILS did not require much tuning to outperform the baseline ILS, we think that tuning enhanced ILS manually is burdensome when one wants to get the best results. Probably, enhanced ILS will require another tuning if the problem instances it encounters in reality have different probability distributions and value intervals (for the task deadlines, workloads and agent capacities) then those we used here. Therefore, an interesting future work would be to incorporate online parameter tuning, to adapt to the problem instance, at hand, on the fly and to spare the user the tedious effort of a manual tuning. In fact, online parameter tuning (such as that which is based on adaptive operator selection [72]) has already shown some success with ILS [72]. Last but not least, we think that tuning ILS's modules online may generate a further enhanced ILS, particularly if we use machine learning – such as Thierens's adaptive pursuit algorithm [72] – to systematically, and at runtime, find the metaheuristic's best modules configuration.

Conclusion

We have introduced a new multi-robot task allocation problem, MRTA-STC, which is relevant for many applications, such as emergency response, goods pickup and delivery, time-critical multi-robot missions in hostile environments and resource allocation such as in Internet of Things. We formulated MRTA-STC as a combinatorial optimization problem which aim is to maximize the number of allocated tasks, under spatio-temporal and capacity constraints on the agents and the tasks.

We formally defined the problem, proved it NP-hard and showed that it is a variant of a well-studied routing and scheduling problem (TOPTW). To tame the problem complexity, particularly in computation restricted and time-critical contexts, we proposed three approaches: (i) a combinatorial auction with suboptimal auction clearing that uses a metaheuristic with random walks [59], (ii) fast and easy-to-implement heuristics which are greedy or insertion-based [73] and (iii) a metaheuristic, enhanced ILS, that is anytime, built upon a stat-of-the-art ILS and has a competitive solution quality while showing a suitable trade-off between solution quality and response time [68]. We experimentally evaluated enhanced ILS (our main approach) against the baseline ILS, on synthetic test instance (which we have generated based on the main benchmark of the related problem). Enhanced ILS performs better then the baseline with regard to solution quality, while showing a prompter response time. Furthermore, we found that enhanced ILS superior performance (relatively to the baseline) is little sensitive to the (free) parameters tuning and it is practically robust to run replication. Finally, enhanced ILS is more utile than the baseline, if we happen to need to interrupt the search at some cutoff time. Another contribution is on how to tune ILS search guidance based on an evaluation against two trivial metaheuristics which we showed how to derive [73]. Having not got time to finish a research work on coalition formation in MRTA, using an insertion-based heuristic and an ILS; we rather contributed to the topic within a survey on MRTA, specifically on coalition-based MRTA ([21], Chapter 2).

In the future, we may extend the problem and the metaheuristic to allocate tasks dynamically. It would be interesting to generalize the problem by prioritizing tasks and decaying their utilities over time. We can improve enhanced ILS with an automated parameters tuning (e.g., with machine learning). As we may automatically configure the metaheuristic, such that to search for the best combination of modules at runtime. Finally, our approaches could further be evaluated on data collected from real scenarios, or on other benchmarks (to elaborate).

Bibliography

[1] NASA Mars Exploration Rovers. https://mars.nasa.gov/mer/mission/. Accessed: 2018-09-3.

[2] Hiroaki Kitano and Tadokoro Satoshi. Robocup rescue: A grand challenge for multi-agent and intelligent systems. AI Magazine, 22(1), 2001.

[3] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. Decentralized coordination in RoboCup rescue. Computer Journal, 53(9):1447–1461, November 2010.

[4] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '14, pages 1495–1503, Richland, SC, USA, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[5] Mark Ashbaugh and Rafael Benguria. The problem of queen dido: Overview of the subject of isoperimetry. https://faculty.math.illinois.edu/~laugesen/dido-isoperimetry-history.pdf. Accessed: 2021-10-02.

[6] Sean Luke. Essentials of Metaheuristics. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[7] Gilbert Laporte. Fifty years of vehicle routing. Transportation Science, 43(4):408–416, November 2009.

[8] Mauro Birattari. Tuning Metaheuristics: A Machine Learning Perspective. Springer, second edition, 2009.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. The MIT Press, third edition, 2009.

[10] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. Introduction to Algorithms. McGraw-Hill Higher Education, second edition, 2001.

[11] Gendreau Michel and Potvin Jean-Yves. Handbook of Metaheuristics. Second edition, 2010.

[12] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science, pages 321–353. Kluwer Academic Publishers, 2002.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

[14] Gendreau Michel and Potvin Jean-Yves. Handbook of Metaheuristics, volume 146 of International Series in Operations Research & Management Science, chapter Tabu Search, pages 41–59. Springer, second edition, 2010.

[15] Marco Dorigo and Thomas Stützle. Ant Colony Optimization. MIT press, 2004.

[16] J.N. Hooker. Testing heuristics: We have it all wrong. Journal of Heuristics, 1:33–42, 1995.

[17] Jiefeng Xu and James P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science, 30:379–393, 1996.

[18] Belarmino Adenso-Díaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research, 54(1):99–114, 2006.

[19] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2, AAAI'07, pages 1152–1157. AAAI Press, 2007.

[20] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, GECCO'02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[21] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. Robotics and Autonomous Systems, 90:55–70, 2017. Special Issue on New Research Frontiers for Intelligent Autonomous Systems.

[22] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research, 32(12):1495–1512, 2013.

[23] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. Coalition formation with spatial and temporal constraints. In Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS), volume 3, pages 1181–1188, 2010.

[24] Paul Scerri, Alessandro Farinelli, Steven Okamoto, and Milind Tambe. Allocating tasks in extreme teams. In Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS), pages 727–734, 2005.

[25] The Centibots project. http://www.ai.sri.com/centibots/. Accessed: 2018-10-12.

[26] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35:254–265, 1987.

[27] Pieter Vansteenwegen and Dirk Van Oudheusden. The mobile tourist guide: An or opportunity. OR Insight, 20(3):21–27, 2007.

[28] Lanah Evers, Ana Isabel Barros, Herman Monsuur, and Albert Wagelmans. Online stochastic uav mission planning with time windows and time-sensitive targets. European Journal of Operational Research, 238(1):348–362, 2014.

[29] Thomas Lemaire, Rachid Alami, and Simon Lacroix. A distributed tasks allocation scheme in multi-uav context. In Proceedings of the 2004 IEEE International Conference on Robotics and Automation, ICRA 2004, April 26 - May 1, 2004, New Orleans, LA, USA, pages 3622–3627, 2004.

[30] B. Coltin and M. Veloso. Online pickup and delivery planning with transfers for mobile robots. In Robotics and Automation (ICRA), 2014 IEEE International Conference on, pages 5786–5791, May 2014.

[31] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. Transportation Science, 39(1):119–139, February 2005.

[32] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. The International Journal of Robotics Research, 23(9):1244–1245, 2004.

[33] Ranjit Nair, Takayuki Ito, Milind Tambe, and Stacy Marsella. Task allocation in the robocup rescue simulation domain: A short note. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, RoboCup 2001: Robot Soccer World Cup V, volume 2377 of Lecture Notes in Computer Science, pages 751–754. Springer Berlin Heidelberg, 2002.

[34] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In Proc. 20th Nat'l Conf. on Artificial Intelligence, AAAI'05, pages 1292–1297. AAAI Press, June 2005.

[35] Paulo Roberto Ferreira, Jr., Fernando Dos Santos, Ana L. Bazzan, Daniel Epstein, and Samuel J. Waskow. Robocup rescue as multiagent task allocation among teams: experiments with task interdependencies. Autonomous Agents and Multi-Agent Systems, 20(3):421–443, May 2010.

[36] M. J. Krieger, J. B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. Nature, 406(6799):992–5, 2000.

[37] Eliseo Ferrante, Ali Emre Turgut, Edgar Duéñez-Guzmán, Marco Dorigo, and Tom Wenseleers. Evolution of self-organized task specialization in robot swarms. PLOS Computational Biology, 11(8), 2015.

[38] Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In Wolfram Burgard and Dan Roth, editors, AAAI, pages 701–706. AAAI Press, 2011.

[39] Xiaoming Zheng and Sven Koenig. Reaction functions for task allocation to cooperative agents. In AAMAS (2), pages 559–566, 2008.

[40] Maitreyi Nanjanath and Maria Gini. Repeated auctions for robust task execution by a robot team. Robotics and Autonomous Systems, 58(9):900–909, 2010.

[41] Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nicholas R. Jennings. Decentralised dynamic task allocation: a practical game theoretic approach. In Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS), pages 915–922, 2009.

[42] Maitreyi Nanjanath. Repeated Auctions for Robust Task Execution by a Robot Team. PhD thesis, University of Minnesota, Minnesota, USA, 2010.

[43] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. The team orienteering problem. European Journal of Operational Research, 88(3):464–474, 1996.

[44] Abilio Lucena. Time-dependent traveling salesman problem - the deliveryman case. Networks, 20(6):753–763, 1990.

[45] Yossi Borenstein, Nazaraf Shah, Edward Tsang, Raphaël Dorne, Abdullah Alsheddy, and Christos Voudouris. On the partitioning of dynamic scheduling problems: Assigning technicians to areas. In Proc. 10th Annual Conference on Genetic and Evolutionary Computation, GECCO '08, pages 1691–1692, New York, NY, USA, 2008. ACM.

[46] Dorota Slawa Mankowska, Frank Meisel, and Christian Bierwirth. The home health care routing and scheduling problem with interdependent services. Health Care Management Science, 17(1):15–30, 2014.

[47] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. Artificial Intelligence, 101(1–2):165 – 200, 1998.

[48] Talal Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, and Andrea Giovannucci. An anytime algorithm for optimal coalition structure generation. J. Artif. Int. Res., 34(1):521–567, April 2009.

[49] José Guerrero and Gabriel Oliver. Multi-robot coalition formation in real-time scenarios. Robotics and Autonomous Systems, 60(10):1295–1307, 2012.

[50] Bruce L. Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. Naval Research Logistics (NRL), 34(3):307–318, 1987.

[51] Dalila Boughaci, Belaid Benhamou, and Habiba Drias. Local search methods for the optimal winner determination problem in combinatorial auctions. J. Math. Model. Algorithms, 9(2):165–180, 2010.

[52] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 22–29. AAAI Press, 2000.

[53] Michael H. Rothkopf, Aleksandar Pekeč, and Ronald Harstad. Computationally manageable combinational auctions. Management Science, 44(8):1131–1147, 1998.

[54] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In IJCAI, pages 548–553. Morgan Kaufmann, 1999.

[55] Dalila Boughaci, Belaïd Benhamou, and Habiba Drias. Une recherche locale stochastique pour le problème de la détermination du gagnant dans les enchères combinatoires. In Gilles Trombettoni, editor, JFPC 2008- Quatrièmes Journées Francophones de Programmation par Contraintes, pages 59–68, Nantes, France, June 2008. LINA - Université de Nantes - Ecole des Mines de Nantes.

[56] Yunsong Guo, Andrew Lim, Brian Rodrigues, and Yi Zhu. Heuristics for a bidding problem. Computers & OR, 33:2179–2188, 2006.

[57] Silvia Suárez, John Collins, and Beatriz López. Improving rescue operation in disasters. approaches about task allocation and re-scheduling. In Proc. PLANSIG 2005, London, UK, 2005.

[58] Magnet framework website. Intelligent Agent for Electronic Commerce. http://www.magnet.cs.umn.edu/. Accessed: 2018-11-01.

[59] Hakim Mitiche and Dalila Boughaci. Combinatorial auctions for task allocation in disaster response. In Proc. Int'l Conf. on Metaheuristics and Nature-inspired Computing (META'12), Sousse, Tunisia, October 2012.

[60] H. H. Hoos and T. Stutzle. Stochastic Local Search: Foundations and Applications. Morgan Kaufumann, 2004.

[61] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. Computers & OR, 36(12):3281–3290, 2009.

[62] Nacima Labadi, Jan Melechovský, and Roberto Wolfler Calvo. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. J. Heuristics, 17(6):729–753, 2011.

[63] Qian Hu and Andrew Lim. An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research, 232(2):276 – 286, 2014.

[64] Mauro Birattari and Marco Dorigo. How to assess and report the performance of a stochastic algorithm on a benchmark problem: mean or best result on a number of runs? Optimization Letters, 1(3):309–311, 2007.

[65] Bruce Hajek. Cooling schedules for optimal annealing. Math. Oper. Res., 13(2):311–329, May 1988.

[66] Kenneth D. Boese and Andrew B. Kahng. Best-so-far vs. where-you-are: Implications for optimal finite-time annealing. Systems and Control Letters, 22:71–78, 1994.

[67] Hakim Mitiche, Julio Godoy, and Maria Gini. On the tuning and evaluation of iterated local search. In Proc. of Metaheuristics International Conference (MIC), Agadir, Morocco, 2015.

[68] Hakim Mitiche, Dalila Boughaci, and Maria Gini. Iterated local search for time-extended multi-robot task allocation with spatio-temporal and capacity constraints. Journal of Intelligent Systems, 28(2):347–360, 2019.

[69] Aldy Gunawan, Hoong Chuin Lau, Pieter Vansteenwegen, and Kun Lu. Well-tuned algorithms for the team orienteering problem with time windows. JORS, 68(8):861–876, 2017.

[70] R Montemanni and LM Gambardella. An ant colony system for team orienteering problems with time windows. Foundations of computing and Decision Sciences, 34:287–306, 2009.

[71] Nacima Labadi, Renata Mansini, Jan Melechovský, and Roberto Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. European Journal of Operational Research, 220(1):15–27, 2012.

[72] Dirk Thierens. Adaptive operator selection for iterated local search. In Thomas Stützle, Mauro Birattari, and Holger. Hoos, editors, Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, volume 5752 of Lecture Notes in Computer Science, pages 140–144. Springer Berlin Heidelberg, 2009.

[73] Hakim Mitiche, Dalil Bougaci, and Maria Gini. Efficient heuristics for a time-extended multi-robot task allocation problem. In International Conference on New Technologies of Information and Telecommunication, Mila, Algeria, 2015. IEEE.