POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Computing Multirobot Paths for Joint Measurements Gathering

Supervisor: Prof. Francesco Amigoni
Co-supervisor: Ing. Jacopo Banfi
Co-supervisor: Ing. Alessandro Riva

Carlo Leone Fanton, 780525

Academic Year 2016-2017

A mia madre.

Abstract

Multirobot systems represent a major sub-field of mobile robotics whose challenges have received a growing attention from researchers in the last few years. Specifically, the problem of performing joint measurements recurs in many robotic applications, like in constructing communication maps from signal strength samples gathered on the field, and in localization and positioning systems.

In this work, we consider an environment represented by a metric graph where a team of robots has to perform a given pre-specified set of joint measurements, which represent the locations where information gathering is needed. The aim of this thesis is to solve one fundamental problem emerging from this scenario: seeking joint paths for the robots to perform all the required measurements at minimum cost.

We prove that the problem of jointly performing measurements from given vertices is NP-hard when either the total traveled distance or the task completion time has to be minimized. Given the difficulty of finding optimal paths in an efficient way, we propose a greedy randomized approach able to cope with both the optimization objectives. Extensive experiments show that our algorithms perform well in practice, also when compared to an ad hoc method taken from the literature.

Sommario

I sistemi multirobot rappresentano un'importante area di ricerca nel campo della robotica, e stanno ricevendo sempre maggiore attenzione dai ricercatori. In particolare, il problema di eseguire misurazioni congiunte ricorre in molte applicazioni, come nella costruzione di mappe di comunicazione derivanti dal rilevamento di campioni di potenza del segnale, oppure nei sistemi di localizzazione e posizionamento.

In questo lavoro, consideriamo un ambiente rappresentato da un grafo metrico in cui una squadra di robot deve eseguire un insieme pre-specificato di misurazioni congiunte, rappresentate dai punti fra i quali è richiesta l'acquisizione di informazioni. Lo scopo di questa tesi è risolvere uno dei problemi fondamentali che emergono da questo scenario: trovare percorsi congiunti per permettere ai robot di eseguire le misurazioni richieste a costo minimo.

Proviamo che il problema di eseguire congiuntamente misurazioni su insiemi di vertici dati è NP-difficile quando dobbiamo minimizzare il totale della distanza percorsa oppure il tempo di completamento. Data la difficoltà di trovare percorsi ottimali in maniera efficiente, proponiamo un approccio greedy randomizzato capace di far fronte alla minimizzazione di entrambi gli obiettivi. Numerosi esperimenti dimostrano che i nostri algoritmi hanno buone prestazioni, anche quando confrontati con metodi ad hoc proposti in letteratura.

Ringraziamenti

Trovare le parole giuste per ringraziare tutte le persone con cui ho la fortuna di condividere la mia vita è più difficile che scrivere lo script Python su cui ho lavorato per mesi, ma ci proverò.

In primo luogo grazie al Professor Amigoni; il suo aiuto preciso, costante e sempre attento mi ha permesso di raggiungere questo importante traguardo. Grazie a Jacopo, presto Dott. Banfi, per il supporto e per i fondamentali consigli che mi ha dato nella stesura di questo elaborato: sarai un super PhD iper ultra swag! Grazie ad Alessandro Riva a.k.a. Alecsh e a tutti gli altri ragazzi dell'AIR Lab con cui sono stato a contatto in questi mesi per le indicazioni e la puntuale assistenza che mi hanno rivolto in alcune fasi del lavoro.

Grazie a mia Mamma per i sacrifici che ha fatto e che continua a fare, e per avermi insegnato che con pazienza e costanza si possono raggiungere grandi obiettivi. Grazie a Irene Orsa, amica, sorella, ziǩrilla e compagna di avventure. Grazie a mio papà che mi ha insegnato che è bello vivere all'insegna della spensieratezza. Grazie a nonna Carla e zio Silvano per la loro affettuosa presenza all'interno del mio percorso di crescita.

Grazie ai miei amici di sempre e per sempre, per tutti i momenti passati e per quelli futuri: Stumpello, Ricky, Simo, Mike, Tommy, Tosca, Tore, Winnie, Marche, e tutti gli altri; siete troppi, la consegna della tesi è alle porte, ma vi ricordo che vi voglio bene anche se non siete in questo elenco.
Grazie a Jari e Vania, così lontani ma così vicini allo stesso tempo, per sopportarmi e supportarmi 24/7.
Grazie ai miei mitici compaesani Fox, Piazza, Omar, Diba e il Conte: Bolladello caput mundi.
Grazie ai miei fidi compagni di studi: Edo e quel sumarèl di Marco. E ovviamente i magici membri della Siusi Gang: Dig, Andre, Cant e Barfe. Un

consiglio: allenatevi a Munchkin, che noia vincere sempre!
Grazie a Caterina per tutto quello che fa per me e per la serenità che mi trasmette, sei un fiore prezioso.

Carlo Leone

Contents

Chapter 1

Introduction

"We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time."

Thomas Stearns Eliot

Multirobot systems (MRSs) represent a major sub-field of mobile robotics whose challenges have received a growing attention from researchers in the last few years [21]. Sensing-constrained planning is central to MRSs. It can be described as the problem of planning the optimal execution of a task where some constraints, like limited range, affect or are imposed to the robot's sensing capabilities. For example, consider the coverage task, where robots are required to sense all the free area of the environment, or patrolling, where robots need to check for the presence of threats at some locations with a given frequency. Although a single-robot system might have a reliable performance, some tasks may be too complex or even impossible for it to accomplish. One of the major challenges for MRSs is to design appropriate coordination strategies between the robots that enable them to perform operations efficiently in terms of time and space. Exploration, surveillance, and target search are domains that exhibit the need for robots to take coordinated decisions accounting for sensing requirements. In these settings, robots typically take sequences of measurements from locations that are determined in order to optimize some objective function related to the traveled distance or to the time taken to complete the task.

In all the cited application domains, robots often need to possess knowledge about the possibility of communicating between pairs of locations of the environment in which they are. For robots that need to cooperate in some tasks, communication is fundamental for exchanging information. Finding pairs of locations where robots can interact and exchange data has therefore primary importance. In order to fulfill this goal, robots often integrate some conservative prior knowledge about communication capabilities (e.g., it is safe to assume full communication within a small distance from a teammate, or if in line-of-sight). An alternative is to build communication models starting from joint signal strength measurements gathered on the field. This task falls within the more general framework of *information gathering.*

Information gathering tasks involve robots taking measurements with the aim of maximizing some cumulative discounted observation value over time [20]. Here, observation value is an abstract measure of reward, which encodes the properties of the robots' sensors, and the spatial and temporal properties of the measured phenomena. Concrete instantiations of this class of problems include monitoring environmental phenomena, disaster response, and patrolling environments to prevent intrusions from attackers.

In this work, we consider a scenario where a team of robots has to perform a given pre-specified set of joint measurements, which represent the locations where information gathering is needed.

The aim of this thesis is to solve one fundamental problem emerging from this scenario: seeking joint paths for robots to perform all the required measurements at minimum cost. The environment is discretized as a graph. Vertices correspond to locations of interest that can be occupied by a single robot, while the weighted edges represent the shortest paths between such locations. On this graph, a subset of edges defines the measurements to be performed. A measure is performed when, at a given time, two robots are placed in the two vertices representing the edge. A tour plan encodes a joint walk for the robots to perform all the required measurements. The team of robots starts the tour from a common location of the environment, the *depot.*

Optimally planning pairwise joint measurements poses additional difficulties with respect to the case in which measurements are performed by single robots. This sensing-constrained planning formulation has received much less attention in the literature than its single-sensing counterpart, even if it can properly comply to many real-world multirobot planning applications. Indeed, optimal solutions might exhibit intricate synchronization patterns, which can be difficult to capture in a systematic algorithmic framework. The problem is approached with the developement of two typologies of algorithms, applicable in the optimization of two different objectives and proposing a *feasible solution* for the *most efficient tour.* A solution is feasible if the tour planned visits every pair of locations of the input set.

The *efficiency* of the tour can be evaluated according to two aspects: distance and time. Therefore the most efficient tour can be computed as the one which minimizes the mission completion time or the one which minimizes the the total cumulative distance the robots travelled.

Previous works reduced this problem to a multirobot graph exploration problem, which was solved for teams of 2 and 3 robots [11]. Minimum cost path computations performed by this approach are known to be extremely inefficient since the complexity is exponential in the number of vertices, and thus heuristic approaches need to be pursued. However, adaptations of standard approaches from the scheduling and sequencing literature [7] do not seem applicable without prohibitive scaling problems. This is the main motivation for studying how to plan optimally pairwise joint measurements from a complexity and approximation point of view, with the objective of identifying and testing a practical resolution approach.

The thesis is structured as follows. Chapter 2 presents an overview of the researches made in the field of multirobot systems which were of inspiration for this study. In Chapter 3 the formalization and detailed description of the problem we address is discussed, with hints on its complexity. Chapter 4 contains the description of the solving methodologies and the explanation of the developed algorithms. Chapter 5 describes the whole system architecture implemented. In Chapter 6 the experiments run to test our framework and compare the performance of the algorithms are shown, along with the obtained results. In Chapter 7, we conclude by summarizing the final evaluations of this thesis and presenting some suggestions for future works.

Chapter 2

State of the Art

This chapter gives an overview about the relevant works in the field of multirobot systems employed in information gathering tasks, underlining their importance. In particular, we focus on joint measurements and on their application in the area of communication maps construction.

Multirobot Systems

In the field of mobile robotics, the study of multirobot systems (MRSs) has grown significantly in size and importance in recent years. Having made great progress in the developement of single-robot control systems, researchers focused their studies on multirobot coordination. One of the major challenges for MRSs is to design appropriate coordination strategies between the robots that enable them to perform operations efficiently in terms of time and working space. Although a single-robot system might display a reliable performance, some tasks (such as spatially separate tasks) may be too complex or even impossible for it to accomplish. As a reference, consider the survey by Yan et al.[21]. Here, several major advantages of using MRSs over single-robot systems are pointed out:

- A MRS has a better spatial distribution.
- A MRS can achieve better overall system performance. The performance metrics could be the total time required to complete a task or the energy consumption of the robots.
- A MRS introduces robustness that can benefit from information sharing among the robots, and fault-tolerance that can benefit from information redundancy.

- A MRS can have a lower cost. Using a number of simple robots can be simpler (to program) and cheaper (to build) than using a single powerful robot (that is complex and expensive) to accomplish a task.

MRSs can be homogeneous (the capabilities of the individual robots are identical) or heterogeneous (the capabilities are different). [21] identified nine primary research topics within the MRS: biological inspiration, communication, architectures, localization, information gathering (including, e.g., exploration and mapping), object transport and manipulation, motion coordination, reconfigurable robots and task allocation.

Among these research topics, the primary focus of this thesis is on employing multirobot systems for a particular type of information gathering task in which the information to be collected is the signal strength between pairs of locations of a known environment, performing a specified set of joint-measurements at minimum cost.

Information Gathering

Multirobot systems have made tremendous improvements in exploration and surveillance. In this kind of problems, robots are required to gather as much information as possible. The system needs to create a complete and accurate view of the situation, which may be used afterwards by some robots to make decisions and perform actions. Therefore, the information gathering system must be able to identify lacking information and take the necessary steps to collect it. As stated in [16], *"developing methods to allow robots to decide how to act and what to communicate is a decision problem under uncertainty"*. Such a system can have many real-world applications. For example, in the aftermath of an earthquake, a team of unmanned aerial vehicles (UAVs) can support first responders by patrolling the skies overhead. By working together, they can supply real-time area monitoring on the movements of crowds and the spread of fires and floods. Teams of UAVs can also be used to track and predict the path of hurricanes [20].

The problem of correctly scheduling information gathering tasks has been approached in different ways. [19] investigates on the need for efficient monitoring of spatio-temporal dynamics in large environmental applications. In this system, robots are the entities in charge of gathering significant information, hence careful coordination of their paths is required in order to maximize the amount of information collected. In this work, a Gaussian Process is used to model the problem of planning informative paths. The amount of information collected between the visited locations and remainder of the space is quantified exploiting the mutual information criterion, defined as the mutual dependence between the entropies of the two variables.

The challenge of vehicles coordinated environment patrolling is addressed in [20]. A near-optimal multirobot algorithm for continuously patrolling such environments is developed deriving a single-robot divide and conquer algorithm which recursively decomposes the graph, until a high-quality path can be computed by a greedy algorithm. It then constructs a patrol by concatenating these paths using dynamic programming.

A game-theoretic attitude about information gathering tasks can be found in [16] where a multirobot model for active information gathering is presented. In this model, robots explore, assess the relevance, update their beliefs, and communicate the appropriate information to relevant robots. To do so, it is proposed a distributed decision process where a robot maintains a belief matrix representing its beliefs and beliefs about the beliefs of the other robots. The decision process uses entropy in a reward function to assess the relevance of their beliefs and the divergence with each other. In doing so, the model allows the derivation of a policy for gathering information to make the entropy low and a communication policy to reduce the divergence.

The contribution of [3] is a fleet of UAVs that must cyclically patrol an environment represented as an unidirect graph where vertices are locations of the environment and edges represents their physical connections. Vertices are divided into two classes: m-type vertices, that robots need to monitor and report, and c-type vertices, from which robots can communicate with the mission control center. Information gathering is performed by defining the delays between successive inspections at locations of interest (m-type vertices), measuring their average latency as the performance metric for a tour, and reporting the information collected to the mission control center. The goal is to compute a joint patrolling strategy that minimizes the communication latencies.

Joint Measurements

In the situations presented above, robots typically take sequences of measurements from locations that are determined in order to optimize some objective function related to the traveled distance or to the time taken to complete the information gathering task. In this thesis, we consider a scenario where a team of robots has to perform a given predefined set of joint measurements in a graph-represented environment. While a measurement is usually defined as a data-acquisition operation performed by a single robot at some location, in this work we consider a joint measurement as a pairwise operation performed by two robots that occupy two different locations at the same time.

A straightforward application of the techniques developed in this thesis is the computation of an optimal schedule for performing joint measurements in the construction of communication maps (see next section). In such a setting, a measurement is performed by two robots at two different locations that exchange some polling data to acquire a signal strength sample.

Localization and positioning systems represent another application domain where joint measurements performed by robots are employed. Robot-to-robot mutual pose estimation can allow robots to estimate their global positions from mutual distance measurements. [22]

Analogous problems can be encountered in the Wireless Sensor Networks (WSNs) field, especially when nodes are mobile units [12]. Examples can be found in multilateration-based settings [8] where optimal sequencing of pairwise measurements can speedup the localization of an external entity, a feature particularly critical when such an entity does not exhibit a cooperative behavior.

Communication and Communication Maps

Communication is a fundamental activity for multirobot systems, as it lies at the basis for the completion of a variety of tasks. Applications like surveillance or search and rescue [5], exploration and environmental monitoring [15], cooperative manipulation, multirobot motion planning, collaborative mapping and formation control [11], heavily rely on sharing knowledge among robots in order to enable informed autonomous decision making. Communication is a central requirement for teams of autonomous mobile robots operating in the real world. In real situations, global communication between robots could be a far too optimistic assumption: that's why robots must build an ad hoc communication network in order to share information and must know about possibility of establishing wireless communication links between arbitrary pairs of locations before moving there [5].

In the literature, this knowledge is called *communication map*. Communication maps provide estimates of the radio signal strength between different locations of the environment and so, they can also be used to predict the presence of communication links. With a reliable communication map, we can *"develop a networks of sensors and robots that can perceive the environment and respond to it, anticipating information needs of the network users,* *repositioning and self-organizing themselves to best acquire and deliver the information*" [11] and plan for multirobot tasks. Thus, the developement of solid communication maps could have a deep impact in many real-world scenarios and this is why many researchers have studied the problem with different approaches.

A first methodology is to estimate if communication between two locations is possible exploiting mathematical formulas. In this case it is not performed any map construction. On the other hand, the strategies adopted so far for building communication maps can be divided in two macro areas:

- *Online construction*: robots do not assume to know the environment in which they are. Therefore they build such maps autonomously with a strategy that guides them during their exploration and data acquisition.

- *Offline construction*: the environment is known; an exploration is not needed and so it can be a priori decided where to send a pair of robots to gather the signal strength between two locations.

Dividing the relevant literature according to the two categories and presenting the main features differentiating one from the other evidences the original contributions of this thesis.

Online Communication Maps Construction

Robotic exploration for communication map building is a fundamental task in which autonomous mobile robots use their onboard sensors to incrementally discover the physical structure of initially unknown environments, before moving to locations where signal samples are gathered. The mainstream approach follows a Next Best View (NBV) process, a repeated greedy selection of the next best observation location, according to an exploration strategy [14]. At each step, a NBV system considers a number of candidate locations between the known free space and the unexplored part of the environment, evaluates them using a utility function, and selects the best one.

In [5] a team of mobile robots has to build such maps autonomously in a robot-to-robot communication setting. The proposed solution models the signal's distribution with a Gaussian Process and exploits different online sensing strategies to coordinate and guide the robots during their data acquisition. These strategies privilege data acquisition in locations that are expected to induce high reductions in the map's uncertainty.

In the online construction, robot teams might have non-homogeneous computational capabilities, a sensitive issue for the computationally-expensive

GP parameter estimation process. This is why two different settings are considered in [5]. In *homogeneous settings* each robot is equipped with sufficient computational power to construct the GP model; in *non-homogeneous setting* only an elite of robots has enough computational power. Both strategies are based on a leader-follower paradigm. Leaders are robots in charge of maintaining a communication map by iteratively estimating the GP parameters that best fit the data acquired so far. They are also in charge of selecting the best locations to be visited in coordination with the corresponding followers.

The previous work is the basis of the approach presented by [15], which proposes a system for a more efficient construction of online communication maps. Here, the number of candidate locations where robots can take measurements is limited by the introduction of a priori communication models that can be built out of the physical map of an environment. In this way the number of candidate locations can be minimized to those that provide some distinctiveness. Also, measurements can be filtered to reduce the Gaussian Process computational complexity, which is $O(n^3)$.

A common problem of [5] and [15] is that a recovery mechanism needs to be adopted if any connection between robots is lost. In order to minimize the eventuality of this risk, [10] studies the problem introducing the concept of periodic connectivity. Specifically, the case in which a mobile network of robots cover an environment while remaining connected is considered. The continual connectivity requirement is relaxed with the introduction of the idea of periodic connectivity, where the network must regain connectivity at a fixed interval. This problem is reduced to the well-studied NP-hard multi-robot informative path planning (MIPP) problem, in which robots must plan paths that best observe the environment, maximizing an objective function that relates to how much information is gained by the robots' paths. Then, is proposed an online algorithm that scales linearly in the number of robots and allows for arbitrary periodic connectivity constraints.

Offline Communication Maps Construction

In an offline setting, the map of the environment is known and therefore an exploration strategy is not needed. The measurements of the radio signal strength between pairs of locations can be executed after having computed which is the most efficient way to perform such measurements. This means planning the shortest tour such that each pair of locations is visited.

The most common approach is representing the map of the environment as a graph. Graph representation is a central point in [11], where the problem of exploration of an environment with known geometry but unknown radio transmission characteristics is formulated as a graph exploration problem. This work presents algorithms allowing small teams of robots to explore two-dimensional workspaces with obstacles in order to obtain a communication map. The proposed implementation reduces the exploration problem to a multirobot graph exploration problem, but algorithmic solutions are only presented for teams of two and three robots.

Heuristic Optimization

In this thesis, the environment is represented by a metric graph and it is proved that the problem is NP-hard when either the total traveled distance or the task completion time has to be minimized. Experiments run on an extensive set of instances show that our algorithms developed to tackle the problem perform well in practice, also when compared against an ad hoc method taken from the literature. The algorithms presented in [11] are not analyzed for what concerns computational complexity. We propose instead a polynomial complexity algorithm which can encompass up to n robots, without specifications on the cardinality limit of the team. The algorithm is developed around the concept of *heuristic optimization* (*HO*). *HO* methods start off with an initial solution, iteratively produce and evaluate new solutions by some generation rule, and eventually report the best solution found during the search process. Heuristic algorithms are designed to solve a problem in a faster and more efficient way than traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed. "*Heuristic algorithms are often used to solve NP-complete problems, a class of decision problems. In these problems, there is no known efficient way to find a solution quickly and accurately although solutions can be verified when given. Heuristic algorithms are most often employed when approximate solutions are sufficient and exact solutions are necessarily computationally expensive*". [13]

The main algorithm's implementation idea derives from the HO method called *Heuristic Biased Stochastic Sampling (HBSS)*. HBSS was designed to solve scheduling and constraint-optimization problems. *"The underlying assumption behind the HBSS approach is that strictly adhering to a search heuristic often does not yield the best solution. Within the HBSS approach, the balance between heuristic adherence and exploration can be controlled according to the confidence one has in the heuristic. By varying this balance, encoded as a bias function, the HBSS approach encompasses a family of search algorithms of which greedy search and completely random search* *are extreme members"* [6]. The HBSS algorithm encompasses a wide spectrum of search techniques that incorporate some mixture of heuristic search and stochastic sampling.

These computational approaches enclose noteworthy concepts better defined in the following chapters.

Chapter 3

Formalization and Complexity of the Problem

3.1 Problem Statement

Let $G = (V, E)$ be a complete graph defined on n vertices. Let $c : E \to \mathbb{Z}^+$ be an edge cost function satisfying the triangle inequality. A team of m robotic agents[1] $A = \{a_1, a_2, \ldots, a_m\}$ is deployed on G. They have homogeneous locomotion capabilities and can move between the vertices of G by traveling along its edges at uniform speed, implying that costs $c(\cdot)$ can represent either distances or traveling times between vertices.

The agents have to perform joint measurements on selected pairs of vertices $M \subseteq E$. A single measurement is considered completed as soon as two agents occupy the pair of vertices at the same time (one agent in each vertex). All the agents start from a common depot $d \in V$ and must come back to it once all measurements have been performed. Since G is a complete metric graph, it can be assumed without loss of generality that each vertex in $V \setminus \{d\}$ will always be part of at least one measurement in M.

The execution of a measurement-gathering task is represented with an *ordered sequence* $S = [s_1, \ldots, s_{|M|}]$ where each element s_k is called *assignment* and associates a pair of agents to a pair of target vertices from which the measurement is performed. During the execution of S, each agent $a_i \in A$ remains still on its current position, until a new vertex is scheduled to a_i by means an assignment. Given S, let us define ${a_i}_k^S$ as the vertex position of the agent $a_i \in A$ after the ordered execution of all the assignments up

[1]*Agent*: autonomous entity which observes through sensors and acts upon an environment, directing its activity towards achieving goals. This word is used as general and formal term for indicating *robots*.

to (and including) the k-th one. These agent positions, initially set to d, capture the evolution of the system while running through S. In accordance with that, the cost of an assignment $s_k \in S$ can be defined as:

$$c(s_k) = \sum_{a_i \in A} c(a_{i\,k-1}^{S}, a_{i\,k}^{S}) \tag{3.1}$$

3.1.1 Constraints

The computation of a solution is subject to the following constraints:

1. We denote with $t_j^S(k)$ the time accumulated by an agent a_j executing the assignments up to (and including) the k-th one. Such a value can be defined recursively. In particular, if the assignment s_k does not schedule any vertex to the agent a_j, then $t_j^S(k)$ remains unchanged and equal to the value assumed in the previous step. Otherwise, the baseline value is increased by:

   $$t(s_k) = \max_{a_i \in A} \; c(a_{i\,k-1}^{S}, a_{i\,k}^{S}).$$

   The above condition captures the presence of waiting times occurring when a robot already occupying a vertex assigned by s_k may be required to wait for a teammate to actually perform the measurement given the previous history of measurements $s_1, \ldots, s_{k-1}$.

2. The final configuration must be equal to the initial one. After visiting all the pairs specified by the M set the agents go back to their starting positions.

3.1.2 Objective Functions

We define a first objective function capturing the *distance cumulatively traveled* by the team of robots and we denote it as SUMDIST$(\cdot)$:

$$\text{SUMDIST}(S) = \sum_{s_k \in S} c(s_k) + \sum_{a_i \in A} c(a_{i\,|M|}^{S}, d). \tag{3.2}$$

A second objective function can be defined as the *mission completion time*, the latest time at which a robot ends its duty arriving at the depot. We call it MAXTIME$(\cdot)$:

$$\text{MAXTIME}(S) = \max_{a_i \in A} \left\{ t_i^S(|M|) + c(a_{i\,|M|}^{S}, d) \right\}. \tag{3.3}$$

A solution encoded as a sequence of assignments may contain some elements whose contribution in the objective function is zero. These zero-cost assignments occur whenever a pair of vertices is measured without changing the positions of the agents on the graph solution.

Incompatibiliy of the two Objectives

The presentation of a simple example can show in practice how the solution encoding works. In doing so, the following will be proved:

Proposition 3.1.2.1. *The* SUMDIST$(\cdot)$ *and* MAXTIME$(\cdot)$ *objectives cannot always be simultaneously optimized, even when* $m = 2$.

Consider the simple graph in Figure 3.1 with $V = \{d, u, v\}$.

Two agents a_1 and a_2 initially placed at the depot d have to perform two

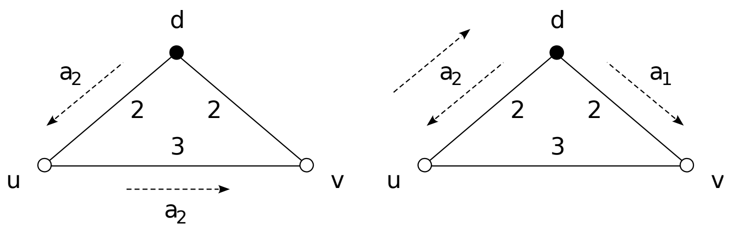


Figure 3.1: Instances of the problem in which the SUMDIST$(\cdot)$ *objective (left) and the* MAXTIME$(\cdot)$ *objective (right) cannot be optimized simultaneously.*

joint measurements defined by the set $M = \{(d, u), (d, v)\}$. By inspection, we can see that a solution S_D^* minimizing the SUMDIST$(\cdot)$ objective is

$$S_D^* = [\langle a_1 \to d, a_2 \to u\rangle, \langle a_1 \to d, a_2 \to v\rangle],$$

with SUMDIST$(S_D^*) = 7$. Here, agent a_1 remains fixed at d while a_2 moves to both u and v, eventually returning at d, with MAXTIME$(S_D^*) = 7$. Focusing on the MAXTIME$(\cdot)$ objective, instead, we see (again by inspection) that an optimal solution S_T^* is

$$S_T^* = [\langle a_1 \to d, a_2 \to u\rangle, \langle a_1 \to v, a_2 \to d\rangle],$$

with MAXTIME$(S_T^*) = 6$ and SUMDIST$(S_T^*) = 8$. To optimize the latter objective, no agent remains fixed at the depot, at the expenses of an increase in the total traveled distance.

3.2 Complexity

NP-Hardness

The previous section defined the two optimization problems we are willing to solve. To prove that they are NP-hard, we use a *reduction* argument. Reducing problem A to another problem B means describing an algorithm to solve problem A under the assumption that an algorithm for problem B already exists. In order to show that our problem is hard, we need to describe an algorithm to solve a different problem, which we already know is hard, using it as a subroutine for our problem. The reduction implies that if problem A was easy, then problem B would be easy too. Equivalently, if problem B is hard, problem A must also be hard. Following this reasoning, if we demonstrate that the decision problems associated to our problems are NP-complete, we prove that the optimization version of our problems is NP-hard.

Suppose to have a solution S. The existence of two values D and T such that SUMDIST(S) $\leq D$ and MAXTIME(S) $\leq T$ can be verified in polynomial time by the algorithms[2] (these two problems are named SUMDIST-D and MAXTIME-T from now on), therefore NP-membership is satisfied. In order to affirm that the two optimization problems related to the minimization of the SUMDIST($\cdot$) and the MAXTIME($\cdot$) objectives are NP-hard, we need to show that the SUMDIST-D and MAXTIME-T are NP-complete. This can be done by providing a reduction from the metric Traveling Salesman Problem (TSP) which is known to be NP-complete [9].

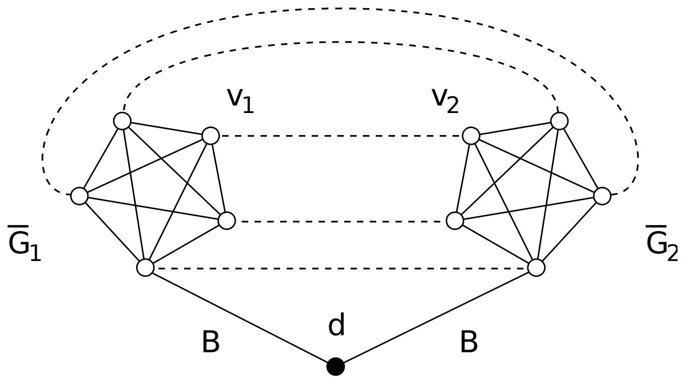


Figure 3.2: A reduction from a metric TSP instance with five vertices.

[2]See Chapter 4.

Let us consider first SUMDIST-D. From a generic instance of metric TSP, we construct a particular instance of SUMDIST-D with 2 robots and a graph $G = (V, E)$ obtained as the metric closure of the graph shown in Figure 3.2. The metric closure of a weighted graph is a complete graph with the same vertices and in which edges are weighted by the shortest path distances between corresponding vertices in the original graph.

In the figure, the original metric TSP graph is replicated twice in two subgraphs $\overline{G}_1$ and $\overline{G}_2$ which are connected to a depot d through the same vertex copy with an edge with cost B. B is defined as the total distance of the Hamiltonian Cycle[3] in G. We denote by v_1, v_2 the two vertices obtained by replicating twice a generic vertex $v \in \overline{V}$. The set of measurements is defined as $M = \{(v_1, v_2) \ \forall v \in \overline{V}\}$, meaning that is composed by all the pairs of vertices copies. Then we set $D = 6B$.

Consider a solution S of SUMDIST-D with SUMDIST(S) $\leq 6B$ in which two robots a_1 and a_2 starting from the depot d need to visit all the vertices of a subgraph. The two robots initially reach $\overline{G}_1$ and $\overline{G}_2$, respectively spending $B + B$, then visit the vertices copies in the order defined by the metric TSP solution spending at most $B + B$, and finally travel back to the depot spending $B + B$. The total cost is not greater than $6B$.

Consider now any solution S in which the measurement associated with the pair of vertices attached to the depot by means of the edge with cost B is not the first one performed. Since $\overline{G}$ (and hence G) is metric, S can always be turned into a solution in which such a measurement is the first one performed without increasing the total solution cost. Therefore, from such a solution, we can immediately derive the existence of a metric TSP solution with total distance at most B by examining the order in which the measurements are made.

The reasoning for the MAXTIME-T problem is the same, but we need to set the initial value of T to $3B$. In this case we need to consider a solution S of MAXTIME-T with MAXTIME(S) $\leq 3B$ where two robots a_1 and a_2, starting from d, firstly reach $\overline{G}_1$ and $\overline{G}_2$ spending B. Then they perform time joint measurements visiting the vertices copies spending at most B, and return to the depot d spending B again. The total cost cannot exceed $3B$.

Since we proved that the decision problems of SUMDIST-D and MAXTIME-T are NP-complete, their optimization version is NP-hard. Also, note that

[3]A Hamiltonian Cycle is a closed loop in a graph that visits each vertex exactly once.

our proof shows that the problems remain NP-hard even on highly-restricted instances, namely, where $m = 2$ and each $v \in V$ appears at most once in M.

Notation and Definitions

Table 3.1 summarizes the relevant terms and definitions presented. Some of them will be encountered in the following chapters.

M	Set of the pairs of points in which a joint measurement is needed. It is given as input for the problem.
Configuration	Given a graph $G = (V, E)$, a configuration is an assignment of m robots to m vertices on the graph.
Assignment	It is a tuple with: the robots that perform a measurement, the pair of points of the M set measured, and the cost of the measurement performed. E.g.: $\langle (a_i, a_j);\ (v_i, v_j);\ c(\cdot) \rangle$ $a_i, a_j \in A$, $v_i, v_j \in M$, $c(\cdot)$ according to the objective function we are minimizing.
Measurement Table	Table that stores the sequence of the assignment costs computed by the algorithms. See Section 4.2.
Move	Calculation of the search strategy of the algorithms that brings from a state to another. See Chapter 4.

Table 3.1: Notation and Definitions.

Chapter 4

Algorithms

In order to tackle the problem introduced in the previous chapter, we implemented four algorithms which can be divided in two classes: *graph exploration algorithms*, that we call *K2* and *K3*, are those presented by [11], and *informed search algorithms* (*Greedy* and *HBSS*[1]) are instead two new algorithms devised for solving our problem.

The first section of this chapter presents the theoretical concepts behind the design of the informed search algorithms, the main contribution of this thesis to the field of study. The second section of the chapter aims at providing an exhaustive and precise explanation of the computations the algorithms perform.

4.1 Problem Framework

The computation of the most efficient tour can be related to a common process used in the field of Artificial Intelligence: *state space search*. The problem is modelled as finding the best sequence of states the problem can be in. Two states are connected if and only if an operation can be performed for shifting from the previous state into the one it is linked to. A state encodes the picture of the world (relevant to the problem) at a certain point along the progression of the plan, while the state space is the universe of all the possible states.
In a state only necessary information is encoded:

- Configuration of the team of robot: where each robot is located on the map.

[1]This algorithm is based on the *heuristic biased stochastic sampling* technique introduced in Chapter 2.

- Measurement table: a table that stores for each robot the measurements performed up to that state. If we are minimizing time, it stores the time needed for each robot to reach the current configuration. If we are minimizing distance, it stores the sum of the distances each robot travelled from the beginning of the tour.
- Points to visit: the list of the pairs of points of the M set not yet visited.

The problem is formulated by specifying 5 elements:

- *Initial state of the problem*: in the initial configuration each robot is located at the depot, the measurement table is set to 0 for all robots, and the M set is complete. No measurement has been performed.
- *Action*(s): given the state s it returns the set of actions that are applicable in s. This means each possible joint measurement actuable from s, considering the pairs of points of M not yet visited.
- *Result*$(s, a) = s'$: is the function that returns the state reached performing action a in state s. The state s' is a successor of s, action a belongs to *Action*(s). Hence, a successor state s' is the state whose configuration is obtained by the application of action a to s.
- $GoalTest(s)$: it is a Boolean function returning *true* if s is a goal state. A goal state corresponds to a state in which every pair of points of the M set has been visited.
- $Cost(s, a)$: is the cost of performing action a in state s. Section 4.2 explains how costs are determined.

A solution to a search problem is a sequence of actions which brings from the initial state to one of the states that satisfy the goal test.

It is convenient to approach this search problem with a *search tree.* This allows us to understand and visualize better the problem. The search tree is a tree where the *root* corresponds to the initial state and the following pattern is repeated:

- Choose a state to expand.
- Apply the goal test to the chosen state.
- If the goal test failed, expand the state.

Search Strategy

The definition of the search tree is not enough. Now we need to institute a procedure for the expansion of the states and search of the solution.

The first consideration to be made is that a complete expansion of the tree is not feasible, because it would lead to an expansion of useless states and, most importantly, taking into account the NP-hardness of the problem[2], would boost the computational complexity of the algorithm. Then, two other factors should be taken into account: *completeness* and *optimality.* A strategy is *complete* if it is always able to find a solution (if a solution exists), and is *optimal* if it is guaranteed to find the minimum cost solution.

The type of search strategy applied in the resolution of the problem could be linked to a typology of search strategy known in literature as *uniform cost*: the possible moves a state s can apply are sorted according to the cumulative costs up to s summed with the cost of the moves. The difference with the traditional strategy is that once a state is chosen, we expand it and consider the states that can be reached only from *that* state. In doing so, we do not keep track of the previously expanded state and so we cannot go back and see what the computation would have been if another action was chosen. This choice is justified by the complexity required by a complete expansion, which would require $O(|M|!)$ steps.

Our search strategy is complete: we will always be able to find a solution because we keep expanding states until each pair of points of the tour is visited.

Since after expanding a state we focus only on state space exploration deriving from that state, this search strategy can't be optimal. To be certain of the optimality of the solution, in fact, a complete search of the state space is needed: this approach though is not feasible in practice, as explained above.

Figure 4.1 shows in practice the behaviour of the Search Strategy adopted.

[2]See Chapter 3.

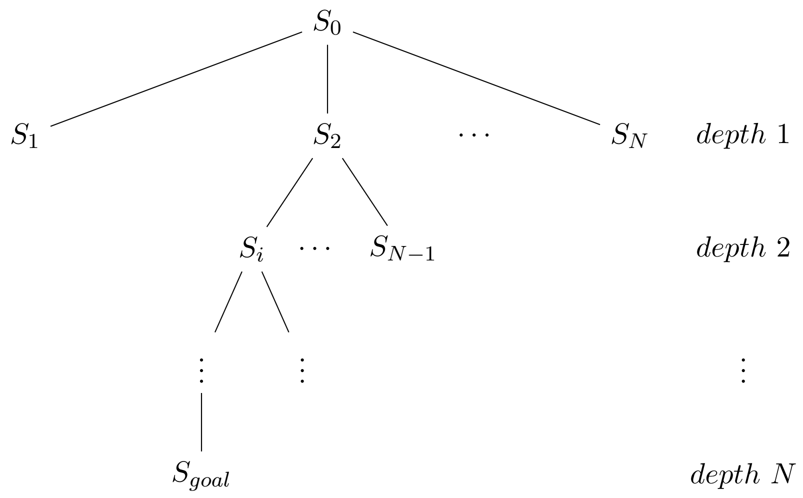


Figure 4.1: Expansion of the search tree of the problem. Starting from the root S_0, the search strategy expands the state chosen until we reach a goal state. Notice that if $|M| = N$, we perform the action of expanding a state N times. Also, the states among which an expansion choice can be made depends on the depth level. Each expansion means that a pair of points of the tour has been visited, so the cardinality of the M set is reduced by 1 everytime a state is chosen. If the depth level is d ($d \geq 1$) it means that we need to consider an expansion among $N - (d - 1)$ states.

4.2 Algorithms

Input Data

Chapter 3 introduced the problem definition and formalization. Here, we recall the elements received as input by the algorithms.

$G = (V, E)$ is a completely connected metric graph defined on n vertices. Each vertex v represents a location reachable by a robot. The team of m robotic agents $A = \langle a_1, a_2, ..., a_m \rangle$ can move between the vertices of G by traveling along edges at uniform speed. The cost of each edge $c : E \to \mathbb{Z}^+$ can represent either distances or traveling times between vertices. Since the graph is metric, the *triangular inequality* holds. The agents have to perform joint measurements on selected pairs of vertices $M \subseteq E$. With the term *configuration* we define an assignment of the m robots to m vertices on the graph. In the initial configuration each robot is assigned to the same starting position, the *depot d*.

Cost of Moves

Before the description of the algorithms, it is essential to accurately define how the *cost of a move* of the search strategy is calculated, since it is the basis for evaluating which joint measurement to perform next. A *move* is an action that brings from a state to another. The cost of a move represents the cost for shifting from a state to another and, therefore, from a configuration to another. In particular, the computation of the cost of a move for two robots a_i and a_j travelling from vertices v_i, v_j to vertices v_k, v_l depends on the objective function we are considering.

- For MAXTIME($\cdot$) the cost of a move is computed as:

$$t = \max\{t_i + c(v_i, v_k), t_j + c(v_j, v_l)\}^3 \tag{4.1}$$

- For SUMDIST($\cdot$) the cost of a move is computed as:

$$d = c(v_i, v_k) + c(v_j, v_l) \tag{4.2}$$

In the MAXTIME($\cdot$) minimization the costs considered are the times needed to reach the vertices, while in the SUMDIST($\cdot$) minimization the costs are the distances between them. Equations 4.1 and 4.2 specify the notation introduced by Equation 3.1.

[3] Recall that t_i and t_j are the costs of the time measurements performed by a_i and a_j up to the state.

As stated in Section 4.1, evaluating moves means taking into account the smallest increase with respect to the objective functions. A practical example will clarify this reasoning.

Suppose to have four robots that after n steps have performed the following measurements. Each value v_i represents the cost incurred by robot a_i with respect to the objective function considered:

Obj. Function	v_1	v_2	v_3	v_4
SUMDIST($\cdot$)	25	17	21	18

Obj. Function	v_1	v_2	v_3	v_4
MAXTIME($\cdot$)	7	7	4	4

Suppose the moves that can be performed are the following (m_{1D} and m_{2D} for the SUMDIST($\cdot$) objective function, m_{1T} and m_{2T} for the MAXTIME($\cdot$)):

Move	Cost			
m_{1D}	5	0	7	0
m_{2D}	0	6	0	8

Move	Cost			
m_{1T}	3	0	2	0
m_{2T}	0	0	1	5

The possible scenarios are:

- SUMDIST($\cdot$): m_{1D} cost is 12, m_{2D} cost is 14. Since the increase brought by m_{1D} in the objective function is lower, m_{1D} is chosen. In the SUMDIST($\cdot$) optimization evaluating moves is simple because the cost depends only on the sum of distances of the considered move.

- MAXTIME($\cdot$): m_{1T} choice results in [10, 7, 10, 4]. m_{2T} choice results in [7, 7, 9, 9]. As outlined in Chapter 3, recall that in pairwise time-measurements a robot already occupying a vertex may be required to wait for a teammate to actually perform the measurement, which is the robot that induces the bottleneck time. That's why in m_{1T} robot a_3 timestamp is set to 10 and in m_{2T} is set to 9.
  Despite the bottleneck time of m_{2T} (5) is higher than m_{1T} (3), m_{2T} is chosen because it reflects a smaller increase in the objective function whose value is 9. With m_{1T} its value would have been 10. As we can see, the evaluation of a move in the case of MAXTIME($\cdot$) optimization is slightly more complex than SUMDIST($\cdot$) because we need to consider the history of the measurements performed by means of the measurement table.

4.2.1 Exploration Graph Algorithms: K2 and K3

After having defined the costs and the relative auxiliary data structures, we start with discussing two algorithms present in the literature [11]. We call these algorithms *K2* and *K3* after the last name of one of the authors of [11]: Vijay Kumar. *K2* stands for the 2-robot case, *K3* for the 3-robot case, as we will see in this chapter. The core idea behind *K2* and *K3* is to compute the most efficient tour with the construction of an auxiliary data structure: the *exploration graph.*

Given a graph $G = (V, E)$ and the M set, an exploration graph $G_k = (V_k, E_k)$ is created such that every vertex in $v \in V_k$ denotes a k-robot configuration on G that measures a subset of M. An edge $e_k^{ij} \in E_k$, exists between two vertices v_k^i, $v_k^j \in V_k$, if the configuration associated to v_k^i is reachable from the configuration associated to v_k^j. Being G a connected graph, k robots can always move from one configuration to another, making G_k complete. The cost assigned to every edge in E_k represents the minimum cost for moving from one configuration to another.

We constructed an exploration graph only for the 2 and 3 robot cases (G_2 and G_3) because for an increasing number of robots it is an approach poorly scalable and performing, since such a procedure would increase the complexity exponentially in the number of agents. The next sections will illustrate how the exploration graph is built and how it is used for computing a solution for the two scenarios.

K2 Algorithm

This algorithm can be used only if we have a team of 2 robots. In such case, *K2* finds the optimal solution. The first step of the algorithm is to compute the vertex set V_2 (Algorithm 1).

Algorithm 1 Vertex set V_2 construction

```
input: environment graph $G = (V, E)$, measurement set $M$
output: a new set of vertices $V_2$
$V_2 = \varnothing$
add $v_2^0$ to $V_2$                    ▷ $v_2^0$ denotes the vertex associated with $d$
for each vertex $v_i$ in $V$ do
    for each vertex $v_j$ in $V$ do
        if $(v_i, v_j) \in M$ then
            $V_2 = V_2 \cup v_2^x$          ▷ $v_2^x$ denotes the vertex associated to $v_{ij}$
```

The main feature of constructing an exploration graph for the 2-robot case is that each vertex $v_2^i \in V_2$ (v_2^0 excluded) corresponds to one of the pair of points of the M set where we want to perform a joint measurement. The weight of each edge $e_2^{ij} \in E_2$ connecting two vertices v_2^i, $v_2^j \in V_2$ is obtained by computing the minimum cost for moving from one vertex to another, which means the minimum cost for performing a measurement from one configuration to another.

Once the exploration graph is obtained, since each vertex $v_2^i \in V_2$ corresponds to a measurement that has to be carried out and since each edge $e_2^{ij} \in E_2$ represents the minimum cost that matches the cost of moving two agents in the respective original problem, we can compute the solution of a TSP[4] on G_2. The transformation outlined above is possible since the presence of exactly two agents univocally identifies the optimal distance functions of the corresponding TSP.

In doing so, we reach an optimal solution for the 2-robot case of the problem.

K3 Algorithm

This algorithm can be used only if we have a team of 3 robots. [11] considered only non-weighted graphs. We extend the approach to cope with weighted graphs. The first step is the computation of the vertex set (Algorithm 2).

Algorithm 2 Vertex set V_3 construction

```
input: environment graph $G = (V, E)$, measurement set $M$
output: a new set of vertices $V_3$
$V_3 = \varnothing$
add $v_3^0$ to $V_3$                         ▷ $v_3^0$ denotes the vertex associated to $d$
for each vertex $v_i$ in $V$ do
    for each vertex $v_j$ in $V$ do
        for each vertex $v_k$ in $V$ do
            if $v_i \neq v_j \neq v_k$ then
                if $(v_i, v_j)$ or $(v_j, v_k)$ or $(v_i, v_k) \in M$ then
                    $V_3 = V_3 \cup v_3^x$        ▷ $v_3^x$ denotes the vertex associated to $v_{ijk}$
```

[4]Traveling Salesman Problem. See Chapter 3.

The vertex set V_3 represents all 3-robot configurations on the original graph $G = (V, E)$ that contain at least one pair of locations where a joint measurement needs to be performed. This means that a pair $v_i, v_j \in M$ might be associated with more than one vertex in V_3 and a TSP cannot be computed. The weight of each edge $e_3^{ij} \in E_3$ connecting two vertices v_3^i, $v_3^j \in V_3$ is obtained by computing the minimum cost for moving from one vertex to another, which means the minimum cost for performing a measurement from one configuration to another.

It's important now to clarify a particular situation that might happen during the execution of *K3*. From the definition of the structure of the exploration graph G_3 follows that in this algorithm robots can move singularly, in pairs or even in triplets. In this way, a robot that moves from a vertex to another might not perform any measurement. Therefore, the measurement table needs to be updated carefully, especially for the MAXTIME($\cdot$) optimization:

- SUMDIST($\cdot$): the measurement table is updated only considering the distance costs resulting from the movement of the robots to their new positions.

- MAXTIME($\cdot$): the measurement table is first updated with the time costs resulting from the movement of the robots to their new position. Then it is updated considering the previous update and the bottleneck times of the robots performing the measurement.

The algorithm computation iterates over the following procedure: at each step the possible configurations reachable from the current one are calculated and sorted. The sorting can be executed with respect to 3 different heuristics:

- *Heuristic O*: moves are sorted according to the cost. Ties are broken considering the minimum increase of the objective function.

- *Heuristic C*: moves are sorted according to how many measurements could be performed. Indeed, the definition of V_3 suggests that moving between two vertices v_3^i to v_3^j implies that more than one measurement might be completed. Ties are broken considering the Heuristic O.

- *Heuristic H*: moves are sorted according to the ratio between the cost and the number of measurements that would be performed. Ties are broken considering the minimum increase of the objective function.

Algorithm 3 *K3* Algorithm

```
input: $G = (V, E)$, initial configuration $v_3^0$, heurisitic $h$, $M$ set
compute $G_3 = (V_3, E_3)$
$v_3 = v_3^0$
while $M$ is not empty do
    $V_{3list}$ = vertices reachable from $v_3$
    sort $V_{3list}$ according to $h$ and pick the first element $v_{3best}$
    delete from $M$ the pairs of points included in $v_{3best}$          ▷ *
    $v_3 = v_{3best}$
```

*: to make this instruction consistent, each heuristic considers only those moves that visit at least one element of the M set.

After sorting, the algorithm performs a *greedy choice*[5] at each step until the M set is empty; the optimal plan for the 3-robot case results in a path over a subset of the vertices in V_3. Preliminary experiments showed that if we want to minimize MAXTIME($\cdot$) the best heuristic to use is H. Instead, if we want to minimize SUMDIST($\cdot$) the best heuristic is O.

4.2.2 Informed Search Algorithms: Greedy and HBSS

The theory behind the developement of these two algorithms is outlined in Section 4.1: the search tree. When a state s is considered, starting from the initial state, three functions are sequentially executed:

1. *EXPAND*: computation of all the possible moves (and their costs) applicable from s.

2. *CHOOSE*: choice of the best move *bm* leading to the best assignment[6], which is selected according to the algorithm considered.

3. *GENERATE*: creation of the state s', generable from s after applying *bm*.

This procedure is repeated until every pair of points of the M set is visited. In each state it is encoded only necessary knowledge[7] for the accomplishment of the operations described above.

[5]A greedy choice is the selection of the element with the smallest incremental increase in the objective function according to the heuristic considered, with ties broken arbitrarily.

[6]See Table 3.1.

[7]See Section 4.1.

These algorithms behave in the same way: in fact, *Greedy* is a special case of *HBSS*. The difference between them is the implementation of the *CHOOSE* function. The general algorithmic structure is described by Algorithm 4.

Algorithm 4 Informed search algorithms

```
input: $G = (V, E)$, initial state $S_0$, objective function $obj$, $M$ set
$STATES = [S_0]$
$S = S_0$
while $M$ is not empty do
    $moves = EXPAND(S)$
    sort $moves$ by minimum increase in $obj$
    $chosen = CHOOSE(moves)$
    $S_{new} = GENERATE(S, chosen)$
    add $S_{new}$ to $STATES$
    delete from $M$ the pair of points included in $S_{new}$
    $S = S_{new}$
```

These algorithms can be run without restrictions on the number of robots composing the team. The next sections delineate a deeper explanation of the issues cited above.

Greedy Algorithm

In a greedy algorithm, the solution is progressively built from scratch. At each iteration, a new element is incorporated into the partial solution under construction, until a feasible solution is obtained and a goal is reached. The selection of the next element to be incorporated is determined by the evaluation of all candidate elements according to a greedy evaluation function. This greedy evaluation function represents the increment in the objective function after adding this new element into the partial solution. In a greedy setting, we select the element with the smallest incremental increase (remember that objective functions are costs, in our application), with ties broken arbitrarily. Costs are evaluated as outlined in Section 4.2.

These properties are embodied within the *Greedy Algorithm* which iteratively expands a state (*EXPAND*) and, after sorting the moves by the minimum increase in the objective function considered, chooses the first element which corresponds to the best one according to the greediness criterion (*CHOOSE*) and generates the corresponding state (*GENERATE*).

The *Greedy Algorithm* is guaranteed to find a feasible solution, since, for this problem, a greedy approach can easily be shown to be complete. However, solutions obtained by greedy algorithms are not necessarily optimal but their performance can boost if we allow the algorithm to explore states in addition to those suggested by the greedy choice. This is the concept behind the implementation of the next algorithm.

HBSS Algorithm

The *HBSS Algorithm* idea is designed around the sampling methodology described in [6]. As mentioned before, a greedy choice throughout the expansions of states is highly probable that does not lead to optimal solutions. A different exploration of the space state, instead, might tend to the discovery of a better solution and, theoretically, even to the optimal one.

We can explore the state space in a more significant way with respect to the greedy choice thanks to *randomization.* Randomization plays an important role in algorithm design, especially when sampling the state space.

Greedy randomized algorithms are based on the same principle guiding pure greedy algorithms but they make use of randomization to build different solutions at different runs. At each iteration, the set of candidate elements is formed by all elements that can be incorporated into the partial solution under construction without violating feasibility. The selection of the next element is determined by the evaluation of all candidate elements according to a greedy evaluation function.

The evaluation function applied in the *HBSS Algorithm* derives from a technique called *heuristic biased stochastic sampling*[8]. This methodology doesn't sample the state space randomly but it focuses its exploration using a given heuristic, which in our case is the greedy one. The adherence to the heuristic depends on another component, the *bias function.* The bias function selection has consequences on the state space exploration depending on the confidence placed in the heuristic: the higher the confidence, the stronger the bias, therefore fewer solutions different from the greedy one are produced. On the other hand, a weaker bias implies a greater degree of exploration since we are not attached to the heuristic; this leads to a higher variety of solutions.

A complete solution is generated starting from the root node and selecting at each step the node sampled by the *HBSS* state space exploration procedure, until all pairs of points in the M set are visited, meaning that

[8]The acronym of the algorithm derives from this name.

a goal node is reached. In order to select a node, the candidate elements are sorted according to the heuristic (e.g., greedy) and a rank is assigned to each candidate, based on the ordering - top rank is 1. The bias function is used to assign a non-negative weight to each candidate based on its rank. After normalization, the normalized weight represents the probability of the candidate of being selected. Notice that each candidate has a non-zero probability of being chosen at each step.

Algorithm 5 shows the pseudocode relative to how the *CHOOSE* function changes with respect to the *Greedy Algorithm*. The general structure regarding *EXPAND* and *GENERATE* functions still holds.

Algorithm 5 *CHOOSE* Function - HBSS Search

```
input: $G = (V, E)$, heuristic function $h$, bias function $b$, state $s$, $M$ set[9]
$children$ = sorting based on $h$ of $moves$
$totalWeight = 0$
for each child in $children$ do
    $rank$[child] = position in $children$                ▷ top rank is 1
    $weight$[child] = $b(rank$[child]$)$
    $totalWeight = totalWeight + weight$[child]
end for
for each child in $children$ do
    $probability$[child] = $weight$[child]$/totalWeight$
end for
$index$ = weighted random selection based on probability
$chosen = children[index]$
```

The *HBSS Search* algorithm will find a feasible solution sampling the state space with the operations depicted above. Notice that the difference with the *Greedy Algorithm* lies in the choice of the move to apply, which might not be the first one in the sorted list (*children*). The given result might be better or worse than the solution found by the greedy one, depending on how we choose the bias function and also by how many outcomes are calculated. In fact, another strength of the heuristic biased stochastic sampling technique is that we can iterate over the number of times we compute a solution through *HBSS Search*.

[9]Even if in this pseudocode some input data are not used, this list is consistent with the function call actually made by the HBBS Algorithm - see Algorithm 6.

HBSS Search runs until a feasible and complete solution is determined. If this process is repeated N times we would obtain a wider exploration of the state space because at each iteration, a complete *HBSS Search* procedure is carried out. With a proper tuning of the bias function, a wide collection of different solutions is likely obtainable and, with an extended range of solutions to consider we have higher probabilities to reach optimality (Algorithm 6).

Algorithm 6 HBSS Algorithm

```
input: $G = (V, E)$, heuristic function $h$, bias function $b$, initial state $S_0$, $M$ set,
number of restarts $N$:
$RESULTS = [\,]$
$HBSS(N, S_0, h, b)$:
for $i$ in range (0, $N$) do
    $res = HBSS\text{-}Search(S_0, h, b)$
    add $res$ to $RESULTS$
end for
```

In *RESULTS* is stored each of the N computations of a complete run of the *HBSS Search* procedure. The minimum value over the instances of the list is the best result calculated.

The iterative sampling performed by the *HBSS* approach depends on the definition of the bias function, that might encompass greedy search and completely random search as extreme members. The bias functions (r is the rank) considered in this work are:

- *logarithmic*: bias(r) = $log^{-1}(r + 1)$
- *polynomial(τ)*: bias(r) = $r^{-\tau}$
- *exponential*: bias(r) = e^{-r}

From preliminary experiments the best bias function typology is the *polynomial*. Chapter 7 will focus on the analysis of the tuning of a proper polynomial degree and of its performance.

Computational Complexity

From the results of Chapter 3 it is clear that, unless P=NP, the existence of polynomial-time optimal algorithms for any of the two objectives is improbable. However, for the special case of a team of two robots, there exist two simple polynomial-time transformations from our optimization problems to particular TSP instances, for which advanced solvers exist [1].

For what concerns the *HBSS Algorithm*, if we assume that an assignment of the team of robots to a configuration can be evaluated in a constant number of steps, each solution can be found in $O(m^2|M|^2 \log m|M|)$ steps, implying that, if the number of runs is polynomial in the input size, the whole algorithm has a polynomial running time. This complexity derives from the fact that for each step in the evaluation of the assignments (the total number of steps before algorithm completion is $|M|$), the cost of each move is computed in $O(m^2|M|)$ and then sorted in $O(\log m|M|)$. Repeating this process $|M|$ times we obtain $O(m^2|M|^2 \log m|M|)$.

Approximation Factor

We now provide a result that could be useful in the study of an approximation factor for the *Greedy Algorithm.*

Let *Algorithm* 1_D be the *Greedy Algorithm* in charge of SUMDIST$(\cdot)$ optimization. We put in relation the maximum cost of a greedy assignment and the cost of an optimal solution.

Let S_D be a solution found by Algorithm 1_D with $M = E$, and let OPT_D be the cost of an optimal solution minimizing the SUMDIST$(\cdot)$ objective function. Then,

$$\max_{s_k \in S_D} \; c(s_k) \leq 2\,\mathrm{OPT}_D\,.$$

Proof. Consider a *single depot multiple traveling salesmen problem* (m-TSP) on G, with m salesmen and a depot d. In this problem, the goal is to find a set of m tours, each starting from d and visiting at least once every vertex in G, such that the sum of the tour costs is minimized. Let C_{mtsp} be the cost of the m-TSP optimal solution. Since in our problem each vertex must be visited at least once, it follows that $C_{\text{mtsp}} \leq \mathrm{OPT}_D$. Then, let C_{mst} be the cost of a minimum spanning tree[10] on G. We have $C_{\text{mst}} \leq C_{\text{mtsp}}$, because from any m-TSP solution a spanning tree can be obtained by removing the last tour edges (the ones returning to the depot). Finally, the cost between

[10]A minimum spanning tree is a least cost tree covering all the vertices in a graph.

any pair of vertices in G cannot be greater than C_{mst}, because of the triangle inequality. Reconstructing the chain of inequalities we have that the sum of the costs of the greedy assignment can never exceed twice OPT_D. $\square$

Chapter 5

System Architecture

5.1 Systems Involved

Computing efficiently multirobot paths for performing joint measurements can be employed in several applications, as already discussed in Chapter 2. In this chapter we describe the system architecture developed for such computation.

We start from a known environment, therefore a physical map is needed. The pairs of points where we would like to perform joint measurements are then chosen on that map and the algorithm, after receiving this information, can start computing the most efficient tour providing the best solution according to the selected objective. This is how the workflow is formed from an abstract point of view; technically the process is more complex and the final output of the algorithms (an m-robots tour) is obtained by the combination of three systems:

- System 1: data acquisition.
- System 2: data processing and solution computation.
- System 3: analysis of the results and experiments on performance of the algorithms.

Figure 5.1 depicts the workflow in which the three systems are involved. The next sections will give a deep explanation of each system's tasks.

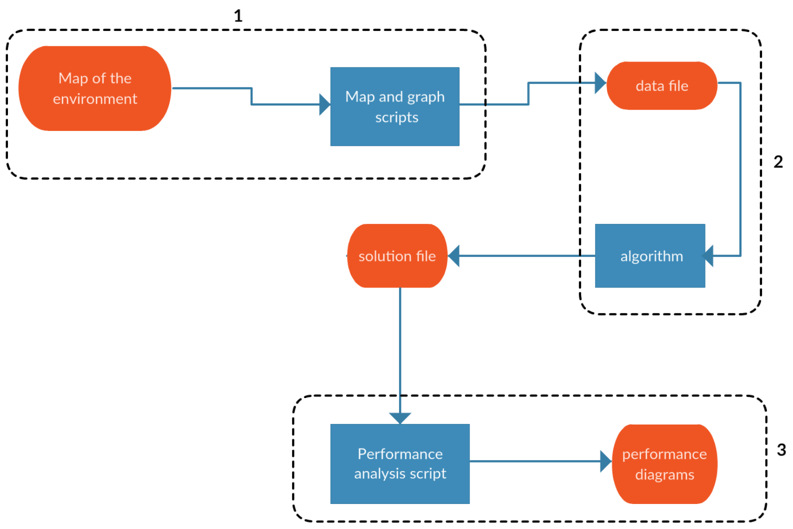
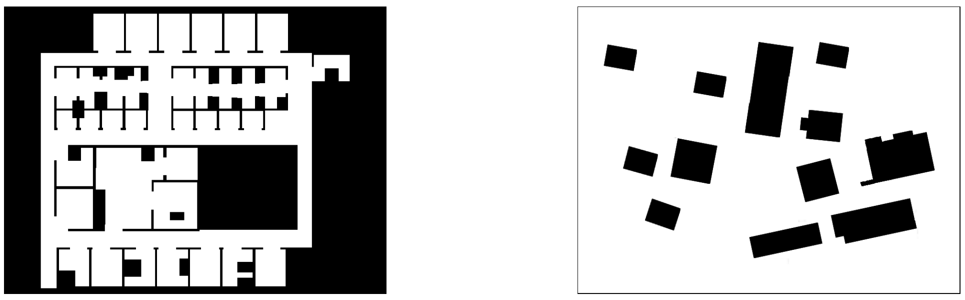


Figure 5.1: Systems interactions.

5.1.1 System 1: Acquiring Data

In this system there are three main entities: *environments, graph scripts,* and *data*. It all starts with a physical map of the environment which is encoded as a 800x600 *.png* file. In this work the environments[1] used were two; from now on they will be referred to as *office* and *open*. These are also the environments which will be used in the experiments of Chapter 6. In the map representations shown in Figure 5.2, robots can move on the *white pixels*, while *black pixels* represents obstacles or unacessible points.



(a) office (b) open

Figure 5.2: Environments - png files.

[1]The environment maps were downloaded from *Radish*, a robotics data set repository. (*radish.sourceforge.net*)

Then, the *create_graph_from_png.py* script discretizes the .png files into a grid of cells. Each vertex v of this grid can be seen as a vertex of a graph. From each vertex v we can derive which are the other vertices it is directly connected to: in a 4-connected grid discretization they are its neighbors along the 4 main directions (at most four). The weight of an arc connecting two vertices v_1 and v_2 is the distance of the shortest path between them. Grid discretization is shown in Figure 5.3.

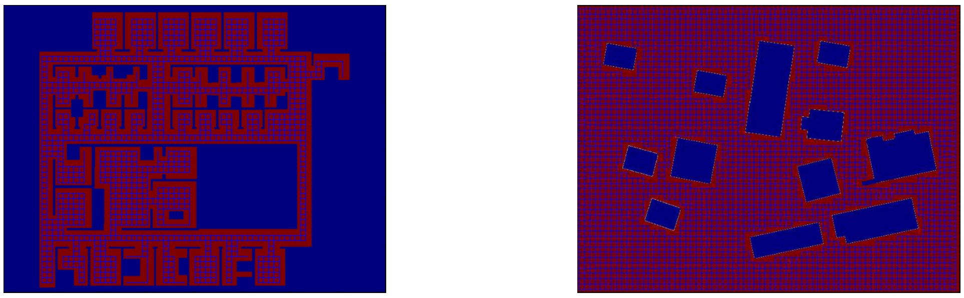
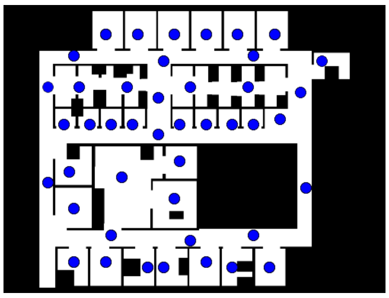

(a) office (b) open

Figure 5.3: Grid of the environments.

In this way, the environment is transformed into a graph. All information about this data structure is stored in a *.graphml* file, which is a comprehensive and easy-to-use file format for graphs. It consists of a language core to describe the structural properties of a graph and of a flexible extension mechanism to add application-specific data. GraphML is based on XML and hence is suited for generating, archiving, or processing graphs.

Now the points of the map that might be visited by robots can be chosen. This task is completed by the *create_ptexplore.py* script, which, after receiving as inputs the .png map of the environment and the underlying graph related to it computed by *create_graph_from_png.py*, allows us to determine the points where a joint measurement might be performed. This is achievable by simply drawing those points on the map: when a point is drawn, it is associated to the closest vertex of the underlying graph.

This information is stored in an *.exp* file. More specifically, the blue points in Figure 5.4 are the vertices of a completed connected graph where the weights of the arcs represents the distances between vertices and are easily computable from a lookup on the .graphml file: it is sufficient to invoke the *shortest_path_dijkstra* procedure between two vertices which is applicable in the GraphML environment.


(a) office

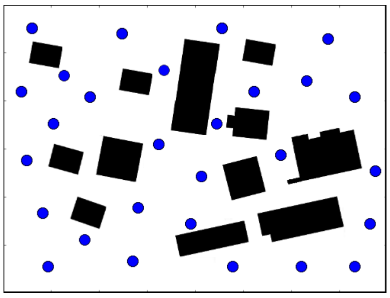

(b) open

Figure 5.4: Environments graphs where joint measurements might be performed.

Notice that we have not yet chosen which are the joint measurements to be done. This is done by the *parsingData.py* script. The elements that belongs to the M set might be picked manually if desired. However, in this work the selection of the elements is done in an automatic way to privilege an exploration whose solution is more significant for a computational evaluation, varying the number of elements in the M set. Recall that the elements belonging to the M set are pairs of locations.

For each environment, 3 different M sets are defined, based on 3 ranges specified in pixels: 250, 500, 1000. The sets are derived in this way: for each vertex v of the graph G specified by the *.exp* file, add to the M set all the pairs formed by v and the other vertices of G whose distance from v is not larger than the range (disregarding obstacles), if the pair has not been already added. This property is summarized by the following equation:
$\forall v_i, v_j \ in \ V, \ v_j \neq v_i :$

$$if \ d(v_i, v_j) \leq range \ and \ (v_i, v_j) \notin M \ \rightarrow \ add \ (v_i, v_j) \ to \ M \tag{5.1}$$

Since the .png map pixel measures are 800x600, with range 1000 all the possible pairs of vertices of the graph need to be visited. This means that with n vertices the maximum cardinality of the M set is:

$$|M| = |E| = \frac{n(n-1)}{2} \tag{5.2}$$

All the interactions between environments files, scripts and data are summarized by Figure 5.5.

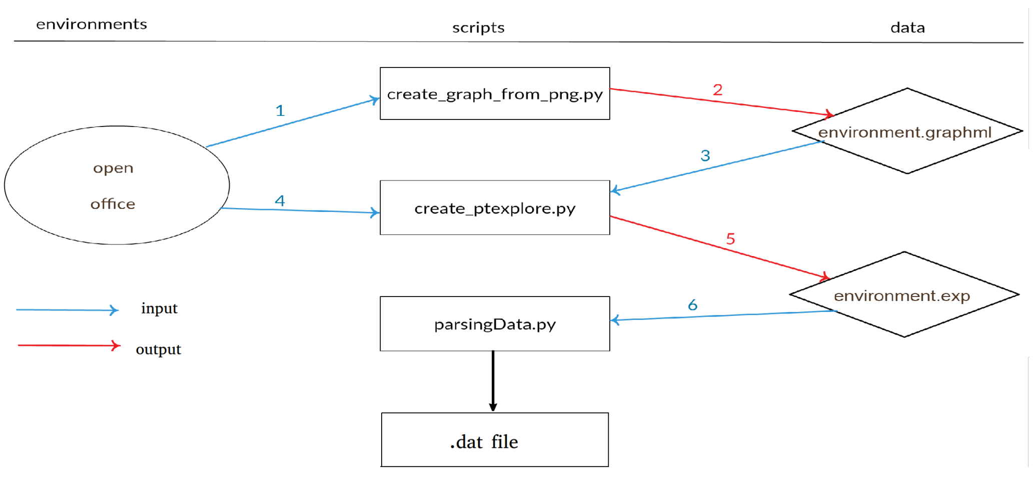


Figure 5.5: Interactions between System 1 entities.

The *parsingData.py* script is the final part of the process of the first system: it creates a *.dat* file which will be the input of the algorithms presented in Chapter 4. The *.dat* file contains all the relevant information needed for a correct computation of the solution. This means:

- *RANGE*: range adopted for the computation of the M set.
- *N_ROBOT*: number of robots of the exploration team.
- *START_POS*: the depot. From here the robots start exploring and here is where they head to after having completed the tour.
  The starting positions chosen for the two environments are depicted in Figure 5.6.

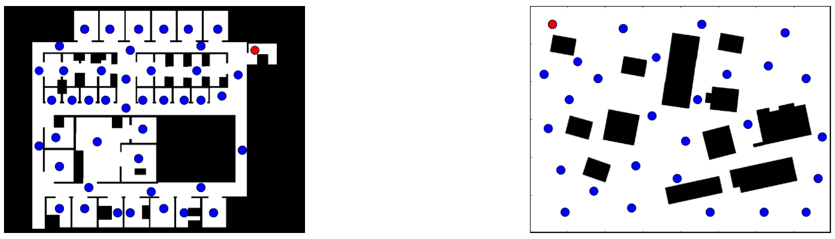


Figure 5.6: Depots in red.

- *N_VERTICES*: number of vertices of the exploring graph. This information eases the parsing process.

- *POINTS_TO_EXPLORE*: the M set.
- *DISTANCE_MATRIX*: this matrix stores, for each vertex v, the distances between v and the other vertices of the graph.
- *VELOCITY*: robots travelling velocity. This data is needed to derive the *time matrix* (the matrix that stores, for each vertex v, the time needed to reach the other vertices of the graph from v) by dividing each element of the *DISTANCE_MATRIX* by this value. In this work *VELOCITY* is always set to 1 m/s to highlight the difference between the computations of the solution according to the two objective functions[2]. Indeed, if $VELOCITY = 1\ m/s$, time and distance matrices are equal.

5.1.2 System 2: Processing the Data and Computing a Solution

The parsing of the *.dat* file is the first step in the computation of the algorithm. In this phase all the significant information and data are saved in proper data structures that will be used to calculate the solution as explained in depth in Chapter 4. The results of the computations made by the algorithms are written in the *solution file* with a layout easy to parse for future analysis. The complete and detailed structure of the output data and other significant information is:

- DATFILE: name of input file with the relevant data needed by the algorithms for the computation of solution.
- ENVIRONMENT: environment considered.
- ALGORITHM: algorithm used for the computation of the solution.
- RANGE: range considered for the creation of the M set.
- STARTING_POS: vertex id of the depot d.
- N_ROBOTS: number of robots of the team.
- SOLUTION: value of the objective function taken into account by the algorithm. It is either the SUMDIST($\cdot$) or the MAXTIME($\cdot$) of the solution.

[2]Chapter 4 explains the difference in the computations of the two objectives. That's why the same matrix leads to different calculations.

- SECONDARY SOLUTION[3] : value of the other objective function. This value is computed updating this *secondary* objective with the assignments calculated for minimizing the *primary* objective.
- COMPUTATIONS[4] : sequence of the solution values computed by the different runs of the *HBSS Algorithm*.
- CONFIGURATIONS: sequence of the configurations in which the robots should be throughout the measurement-collecting task.
- ROBOT_MEASURING: sequence of the id of robots performing a joint measurement. (e.g., $\langle (a_2, a_7); (a_3, a_5); ... \rangle$, if $m \geq 7$)
- MEASUREMENT TABLE: sequence of assignment costs for each robot.
- EXECUTION TIME: execution time (in seconds) of the algorithm.

5.1.3 System 3: Analysis of the Results and Experiments on Performance of the Algorithms

System 3 is in charge to analyze the results saved in the solution files, comparing the algorithms' performance and the solutions obtained.

Before running the experiments, the solution files are parsed by the *analysis scripts* and the valuable data for empirical settings[5] are extracted and stored in a nested *python dictionary*: this specific data structure is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a $\langle key, value \rangle$ pair. Keys are unique and python dictionaries are optimized to retrieve values when the key is known. In this way, accessing values is really simple. The structure of the developed dictionary and related keys is:

$$[environment] \rightarrow [algorithm] \rightarrow [range] \rightarrow [robot\ number] \rightarrow [objective] \rightarrow [tau]$$

The *tau* key was defined only for the *HBSS Algorithm*. The arrows specify the order of data access of the dictionary: at first, we can retrieve the data related to a specific environment. Then, we can retrieve data related to the computations made by a specific algorithm in that environment. Then, we can filter those data by range, and so on.

[3]Only for *Greedy Algorithm*.

[4]Only for *HBSS Algorithm*.

[5]These data are: environment, algorithm, range, number of robots, objective, tau degree (only for *HBSS*).

After parsing the solution files, the significant data stored in the dictionary are extracted and used to plot different diagrams for assessing considerations on the algorithms' performance. The structures of the diagrams has been implemented ad hoc thanks to *matplotlib*[6], a Python plotting 2D library which can produce plots, histograms, scatterplots, etc. Implementation and results of such diagrams are covered in depth in Chapter 6.

[6] *https://matplotlib.org*

Chapter 6

Experiments

In order to evaluate the performance of the proposed algorithms in practical settings, we consider the environments *office* and *open* illustrated in Chapter 5. As depicted in that chapter, vertices on positions of interest (e.g., rooms in the office) were manually created over the bitmaps and edge costs were obtained as shortest paths among pairs of vertices. The number of robots composing the team can vary from 2 to 10 throughout the experiments.
The set of target measurements M is populated according to the distance-based criterion outlined in Equation (5.1). The cardinalities of M are reported in Table 6.1.:

px range	**open**	**office**
250	97	382
500	280	881
1000	378	946

Table 6.1: The cardinalities of the M set.

Three sets of experiments were run to analyze the performance of the algorithms and relative solutions under different points of view:

1. *τ tuning*: investigation on the impact of the choice of the τ parameter of the bias function used in *HBSS*[1].
2. *Algorithms comparison*: examination of the different performance of the algorithms.
3. *Greedy objective comparison*: observation on the results of the *Greedy Algorithm* computation in the minimization of both objectives.

[1]*polynomial(τ)*: bias(r) $= r^{-\tau}$

6.1 Tuning of the τ Parameter

In the first set of experiments, we investigate the impact of the choice of the polynomial degree of the bias function in the solutions found by the *HBSS Algorithm*. The value of this parameter influences the quality of the outcomes computed, because the bias function relies on it when assigning a weight (and hence a probability) to the elements among which an expansion of the state space should be considered.

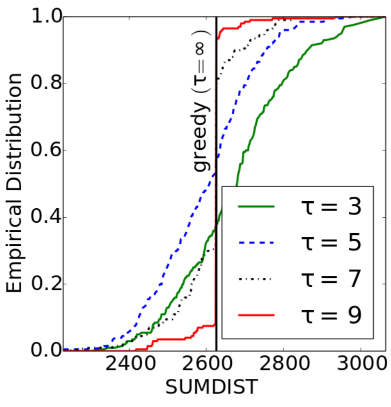
For fulfilling this purpose, we fix the number of robots and the range to intermediate values: 5 robots and 500 px. Figure 6.1 shows the distributions of the solution costs, obtained in both the environments, for $\tau \in \{3, 5, 7, 9\}$ and over 200 restarts. Each curve represents the probability of finding a solution cost less than or equal to a certain value, at each run of the corresponding algorithm. For example, in the *open* environment, the probability that the algorithm for $\tau = 5$ finds a solution S, such that $\text{SUMDIST}(S) \leq 2600$, is between 0.4 and 0.5 at each run. Although the set of tested values is limited, it is fairly easy to infer the general trend of the curves with respect to the τ parameter: increasing τ the distribution tends to fit the step distribution of a pure greedy algorithm. This is quite predictable, thinking about the parameter meaning: the *Greedy Algorithm* can be seen as a special case of *HBSS* with $\tau = \infty$.

Using the pure greedy performance as reference, the results highlight that the curves of $\tau = 3$ and $\tau = 7$ are dominated in all the four scenarios. In particular, $\tau = 5$ achieves overall good performance, while $\tau = 9$ behaves generally bad.

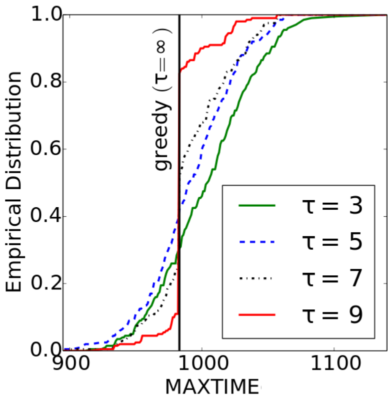
Figure 6.2 depicts the empirical distributions of the solution costs found by *HBSS* w.r.t. $\text{SUMDIST}(\cdot)$ and $\text{MAXTIME}(\cdot)$ minimization, in the *open* environment, with 250 px range. The number of robots m is set to 5 and 8, and again, $\tau = 5$ dominates $\tau = 3$ and $\tau = 7$ in all the scenarios, with $\tau = 9$ being the worst.

In Figure 6.3 we show the results of two different environments with two different ranges, because the cardinalities of their respective M set is almost equal (see Table 6.1). The performance is similar even with a large number of robots, confirming the considerations made.

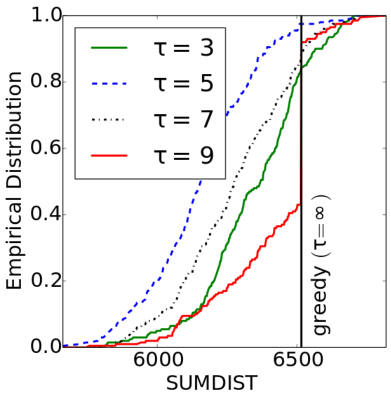
Figure 6.1: Empirical distributions of the solution costs found by HBSS w.r.t. SUMDIST$(\cdot)$ *and* MAXTIME$(\cdot)$ *minimization with 5 robots and 500 px range.*



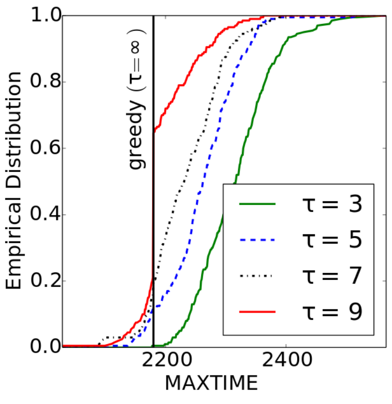
(a) open



(b) open

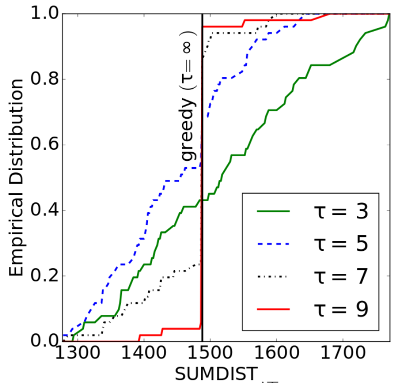


(c) office

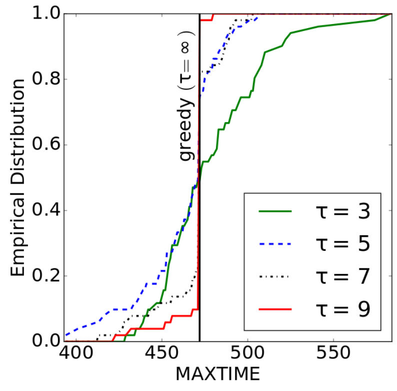


(d) office

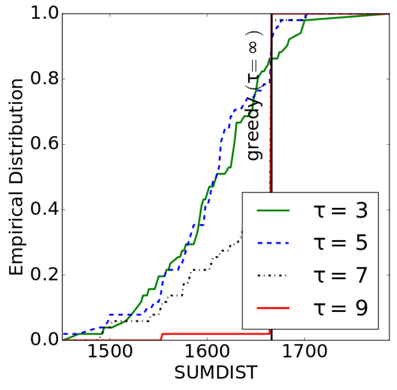
Figure 6.2: Empirical distributions of the solution costs found by HBSS w.r.t. $\mathrm{SUMDIST}(\cdot)$ *and* $\mathrm{MAXTIME}(\cdot)$ *minimization, open environment, 250 px range.*



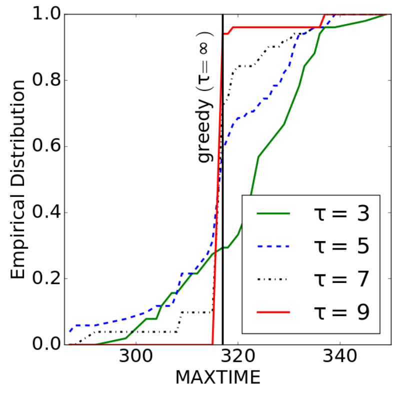
(a) 5 robots



(b) 5 robots

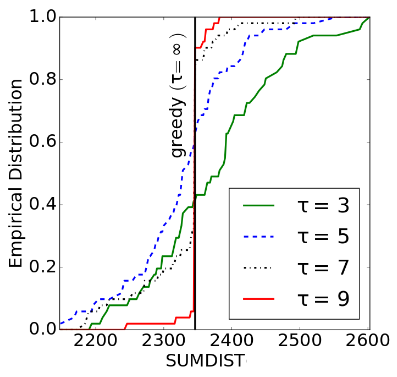


(c) 8 robots

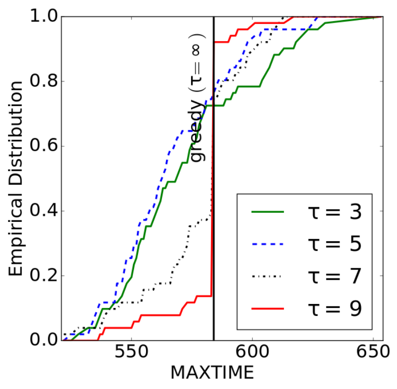


(d) 8 robots

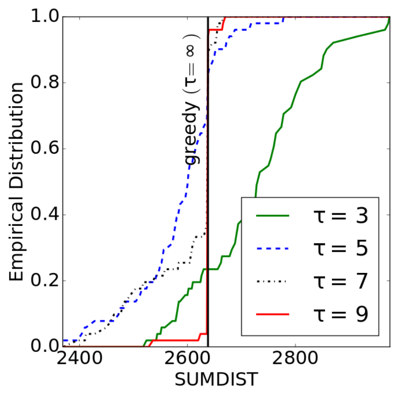
Figure 6.3: Empirical distributions of the solution costs found by HBSS w.r.t. $\mathrm{SUMDIST}(\cdot)$ *and* $\mathrm{MAXTIME}(\cdot)$ *minimization. The number of robots is* $m = 10$.



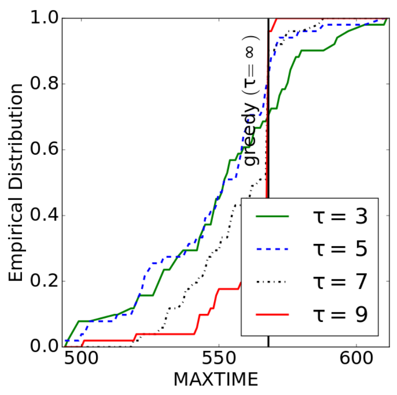
(a) open - range 1000.



(b) open - range 1000.



(c) office - range 250.



(d) office - range 250.

6.2 Algorithms Comparison

In the second set of experiments, we vary the number of robots and we compare the results obtained by the *Greedy Algorithm* ($\tau = \infty$) to those obtained by *HBSS*. The previous set of experiments pointed out that, in general, a good value for the parameter τ of the bias function of the *HBSS Algorithm* is $\tau = 5$. The number of restarts for each setting was fixed to 50. The optimal solution cost is computed only for $m = 2$ robots with *K2*, exploiting the transformation described in Chapter 3. With $m = 3$ robots, the algorithm *K3* is also used.

In Table 6.2, the costs of the solutions found by all the algorithms are shown for the *open* environment. The *HBSS* with $\tau = 5$ outperforms the pure greedy version in all the problem instances. This, of course, comes at the expense of greater computational times. The results obtained by *HBSS* are comparable even with those obtained by the optimal algorithm *K2* for $m = 2$. The gap could be further reduced by increasing the number of restarts. In case of $m = 3$ robots, the solution costs obtained by $\tau = \infty$ are very similar to those of solutions obtained by *K3*, since they both operate greedy choices. However, it is outperformed by *HBSS* with $\tau = 5$ in all the considered instances.

For the SUMDIST($\cdot$) objective and for range 250 px it is interesting to notice that the solution cost does not always decrease, as the number of robots increases. This is due to the greedy nature of the algorithm, which does not take into account the final returns to the depot. The same behavior seems to appear for range 500 px and large numbers of robots, suggesting the presence of a local minimum. As a consequence, the minimization of the total traveled distance by means of greedy algorithms requires additional attention in the selection of the number of robots.

In Table 6.3, we show the costs of the solutions found by all the algorithms in the *office* environment. Again, *HBSS* with $\tau = 5$ outperforms the pure greedy version in all the problem instances considered. Some data are missing, due to the following reasons:

- With range 500 px, we focused only on $m = 5$ for tuning the τ parameter. Results with $m = 2$ and $m = 3$ were computed for a comparison with *K2* and *K3*.
- With range 1000 px, some calculations were stopped because they exceeded the timeout threshold. This is a consequence of the cardinality of the M set given as input (see Table 6.1).

- With ranges 500 and 1000 px, *K2* was not computed for both objectives because the python library[2] in charge of calculating the TSP could not perform computations on such input sizes (881 and 946 points for 500 and 1000 px respectively - see Table 6.1).

	HBSS Algorithm for SUMDIST(·)						*HBSS Algorithm for* MAXTIME(·)					
	Range: 250 px		Range: 500 px		Range: 1000 px		Range: 250 px		Range: 500 px		Range: 1000 px	
Robots (m)	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$
2	1710	1458	3970	3902	5074	5020	1368	1350	3745	3656	4793	4760
3	1658	**1362**	3284	**3264**	4418	**4138**	814	**729**	1922	**1865**	2491	**2416**
4	1520	**1296**	3108	**2714**	3444	**3274**	589	**544**	1298	**1224**	1662	**1598**
5	1488	**1278**	2626	**2356**	2892	**2840**	472	**394**	983	**924**	1228	**1159**
6	1398	**1344**	2274	**2036**	2636	**2516**	380	**339**	791	**712**	914	**871**
7	1400	**1360**	2168	**1932**	2430	**2328**	379	**318**	656	**593**	857	**739**
8	1666	**1452**	2124	**1928**	2326	**2188**	317	**287**	568	**524**	729	**653**
9	1600	**1542**	2078	**1950**	2300	**2166**	311	**285**	475	**465**	634	**565**
10	1686	**1596**	2014	**1902**	2346	**2148**	286	**278**	467	**424**	584	**525**
	Optimal Algorithm *K2* for SUMDIST(·)						Optimal Algorithm *K2* for MAXTIME(·)					
2	**1346**		**3498**		**4546**		**1263**		**3433**		**4506**	
	K3 Algorithm for SUMDIST(·)						*K3* Algorithm for MAXTIME(·)					
3	1552		3524		4358		1022		1897		2484	

Table 6.2: Costs of solutions found by the algorithms in the open environment. Bold indicates the best results.

[2] *Gurobi.py*, the TSP solver coming with each GUROBI installation. GUROBI is an optimization solver for mathematical optimization and operations research.

HBSS Algorithm for SUMDIST(·)

	Range: 250 px		Range: 500 px		Range: 1000 px	
Robots (m)	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$
2	4472	4160	9038	**9009**	9532	**9455**
3	4268	**3706**	8078	**7960**	8684	**8422**
4	3780	**3278**	6888	-	7552	**7036**
5	3342	**2876**	6516	**5664**	6618	**6088**
6	3220	**2704**	5528	-	5682	**5258**
7	2696	**2380**	5090	-	5042	**4628**
8	2668	**2310**	4234	-	4252	**4190**
9	2742	**2362**	4266	-	4500	-
10	2638	**2370**	4116	-	4136	-

HBSS Algorithm for MAXTIME(·)

	Range: 250 px		Range: 500 px		Range: 1000 px	
Robots (m)	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$	$\tau = \infty$	$\tau = 5$
2	3966	3902	8632	**8504**	9369	**9276**
3	2214	**2112**	4613	**4477**	4909	**4784**
4	1569	**1426**	3082	-	3260	**3123**
5	1138	**1041**	2180	**2110**	2276	**2250**
6	875	**810**	1710	-	1922	-
7	790	**699**	1370	-	1561	-
8	630	**607**	1274	-	1359	-
9	593	**532**	1176	-	1087	-
10	568	**493**	904	-	985	-

Optimal Algorithm *K2* for SUMDIST(·)

2	**3626**	-	-

Optimal Algorithm *K2* for MAXTIME(·)

2	**3498**	-	-

K3 Algorithm for SUMDIST(·)

3	4002	8292	8872

K3 Algorithm for MAXTIME(·)

3	2242	4698	4941

Table 6.3: Costs of solutions found by the algorithms in the office environment. Bold indicates the best results.

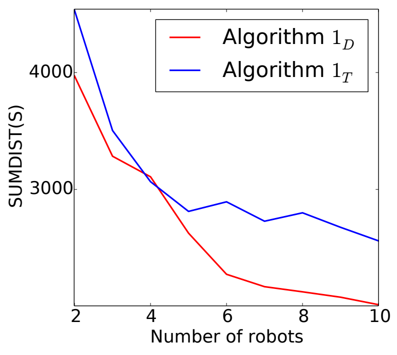
6.3 Greedy Objective Comparison

In the last set of experiments, we investigate the practical dependency between the optimization of our objectives, after having established, with Proposition 3.1.2.1, that in general both the objectives cannot be optimized at the same time. Figures 6.4, 6.5, and 6.6 show the solutions found by the greedy algorithms, evaluated according to the same objective. Specifically, we run the *Greedy Algorithm* and measure the quality of the solutions produced according to *both* SUMDIST(·) and MAXTIME(·) objectives in the *open* and *office* environments. In the legend, *Agorithm* 1_D is the *Greedy Algorithm* that computes the solution minimizing SUMDIST(·), *Agorithm* 1_T is the one which minimizes MAXTIME(·). A comparison is possible thanks to the way the *Greedy Algorithm* has been implemented. Recall that, at each step, when the algorithm updates the value of the objective function it is minimizing, it also updates the value of the other objective function with the configurations computed according to the minimization of the former one.

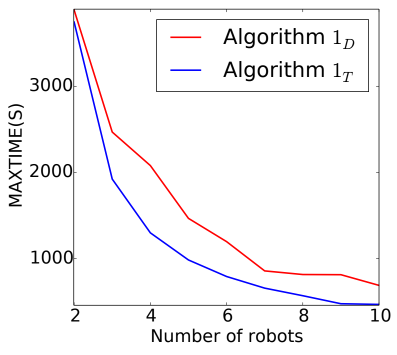
Even if the algorithms perform generally better in their respective objectives (as largely expected), the difference is relatively small. In particular, in case of the MAXTIME(·) objective the gap remains limited even when increasing the number of robots. This suggests that, in general, it may be possible to obtain good solutions with respect to both the objectives.

Figure 6.4: Solutions found by the pure greedy algorithms, evaluated according to both the objectives, with 500 px range.

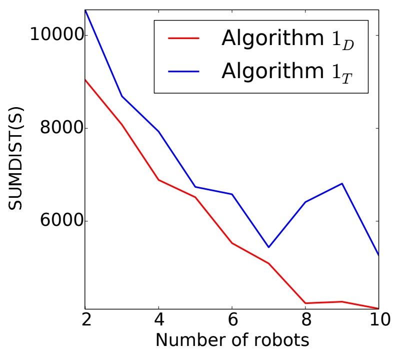
(a) open



(b) open



(c) office



(d) office

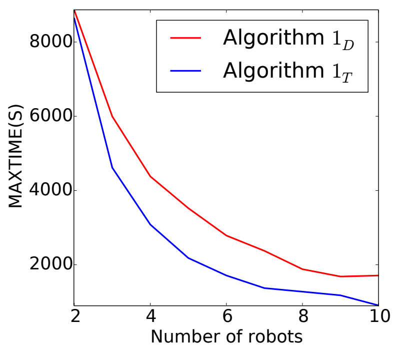


Figure 6.5: Solutions found by the pure greedy algorithms, evaluated according to both the objectives, with 1000 px range.

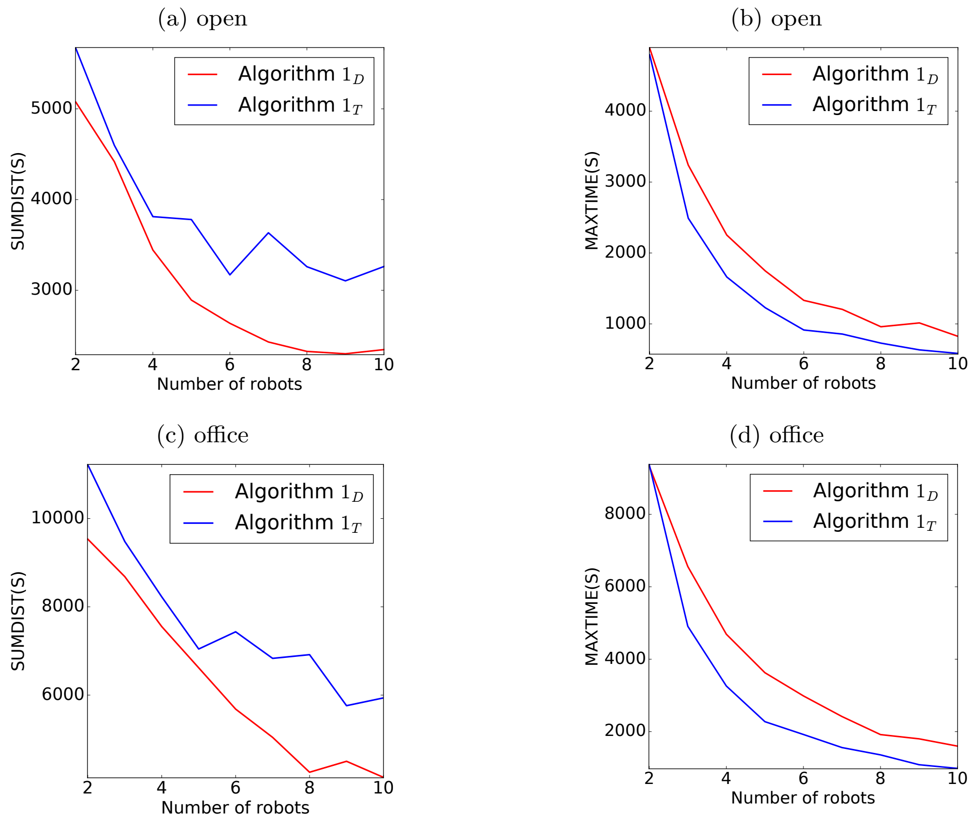
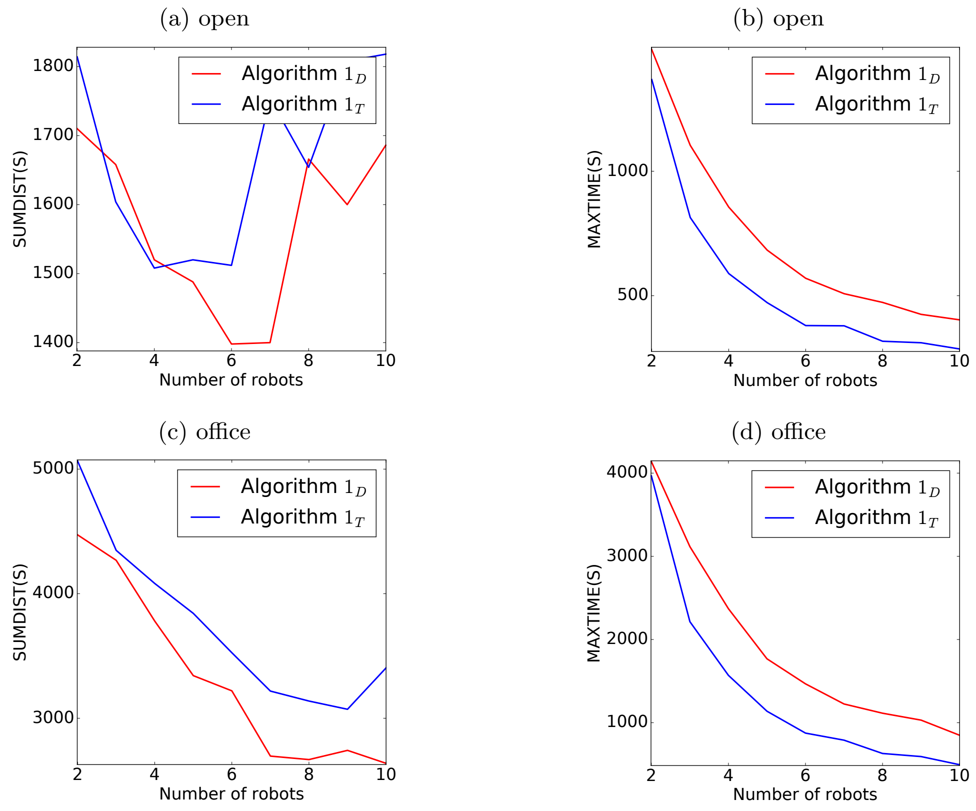


Figure 6.6: Solutions found by the pure greedy algorithms, evaluated according to both the objectives, with 250 px range.



Chapter 7

Conclusions

This thesis investigated a multirobot information gathering task in which robots must efficiently visit pairs of locations of a known environment. We focused on the developement of proper methods for calculating the most efficient tours for the robots to visit a set of pairs of locations given as input for the problem.

Specifically, this thesis proposed a systematic study of a graph-theoretical approach for computing a set of robot paths allowing to efficiently perform pairwise joint measurements, an open issue in the multirobot systems literature. After the formalization of the problem and the proof of its NP-hardness complexity, a resolution methodology based on state space search was presented and then implemented providing efficient algorithms suitable for practical settings. Given the difficulty of finding optimal paths, we proposed a greedy algorithm, together with its randomized version, able to deal with the optimization of both objectives: minimization of the total travelled distance and minimization of the task completion time.

Extensive experiments, including a significant phase devoted to parameter tuning, were run on two environments in order to assess the characteristics of the algorithms. A wide number of observations on these experiments and a deep analysis of the results confirmed the overall good performance of the solving strategies implemented.

From a practical standpoint, in future it could be worth focusing on a particular robotic application requiring to perform joint measurements and studying the relationship between online approaches and the proposed offline resolution scheme. For example, in a context of communication map building where the environment is fully known in advance, it would be interesting to compare the quality of the maps obtained with the online approach of

[5] and [15] to the quality of those obtained with the help of the algorithms presented in this thesis. It is often difficult to assess how much room for improvement online exploration strategies have [14]. A complete evaluation should include a comparison between online and offline performance, based on the competitive ratio. The competitive ratio of an online algorithm a is $\frac{P_{on}}{P_{off}}$, where P_{on} is the performance of a in an environment and P_{off} is the performance of the optimal offline algorithm that knows the environment in advance. While modelling the signal's distribution with a Gaussian Process [5], a priori communication model usage [15], and periodic-connectivity network implementation [10] have been addressed using online strategies, for *offline* communication maps construction a theory supporting efficient algorithms has not been yet developed and ad hoc methods are usually employed. Focusing on the developement of appropriate strategies is the first step to assess such comparison and it was one of the central motivations of this work.

Bibliography

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. *The traveling salesman problem: a computational study.* Princeton University Press, 2011.

[2] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Princeton University Press, 2006.

[3] J. Banfi, N. Basilico, and F. Amigoni. *Minimizing communication latency in multirobot situation-aware patrolling.* In Proc. IROS, pages 616–622, 2015.

[4] J. Banfi, A. Quattrini Li, N. Basilico, F. Amigoni, and I. Rekleitis. *Asinchronous multirobot exploration under recurrent connectivity constraints.* In Proc. ICRA, pages 5491–5498, 2016.

[5] J. Banfi, A. Quattrini Li, N. Basilico, I. Rekleitis, and F. Amigoni. *Multirobot Online Construction of Communication Maps.* In Proc. ICRA, pages 2577–2583, 2017.

[6] J. Bresina. *Heuristic-biased stochastic sampling.* In Proc. AAAI, pages 271–278, 1996.

[7] P. Brucker and P. Brucker. *Scheduling algorithms*, volume 3. Springer, 2007.

[8] S. Capkun and J. Hubaux. *Secure positioning in wireless networks.* IEEE J Sel Area Comm, 24(2):221–232, 2006.

[9] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[10] G. Hollinger and S. Singh. *Multirobot coordination with periodic connectivity: theory and experiments.* IEEE T Robot, 28(4):967–973, 2012.

[11] M. Hsieh, V. Kumar, and C. Taylor. *Constructing Radio Signal Strength Maps with Multiple Robots.* In Proc. ICRA, pages 4184–4189, 2004.

[12] L. Hu and D. Evans. *Localization for mobile sensor networks.* In Proc. MobiCom, pages 45–57, 2004.

[13] V. Kenny, M. Nathal, and S. Saldana. Heuristic algorithms. `https://optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms`. Accessed 2017-11-23.

[14] A. Quattrini Li, F. Amigoni, and N. Basilico. *Searching for Optimal Off-Line Exploration Paths in Grid Environments for a Robot with Limited Visibility.* In Proc. AAAI, pages 2060–2066, 2012.

[15] P. Krishna Penumarthi, A. Quattrini Li, J. Banfi, N. Basilico, F. Amigoni, J. O'Kane, I. Rekleitis, and S. Nelakuditi. *Multirobot Exploration for Building Communication Maps with Prior from Communication Models.* In Proc. MRS, 2017.

[16] J. Renoux, A. Mouaddib, and S. Le Gloannec. *A Distributed Decision-Theoretic Model for Multiagent Active Information Gathering.* In Proc. MDAI, 2014.

[17] M. Resende and C. Ribeiro. *GRASP: Greedy Randomized Adaptive Search Procedures.* Kluwer Academic Publishers, 1995.

[18] A. Riva and F. Amigoni. *A GRASP Metaheuristic for the Coverage of Grid Environments with Limited-Footprint Tools* . In Proc. AAMAS., pages 484–491, 2017.

[19] A. Singh, A. Krause, C. Guestrin, and W. Kaiser. *Efficient Informative Sensing using Multiple Robots.* J Artif Intell Res, 34:707–755, 2009.

[20] R. Stranders, E. Munoz de Cote, A. Rogers, and N. Jennings. *Near-optimal continuous patrolling with teams of mobile information gathering agents.* AIJ, 195:63–105, 2012.

[21] Z. Yan, N. Jouandeau, and A. Ali Cherif. *A Survey and Analysis of Multi-Robot Coordination.* Int J Adv Robot Sys, 10(12):1–18, 2013.

[22] X. Zhou and S. Roumeliotis. *Robot-to-robot relative pose estimation from range measurements.* IEEE T Robot, 24(6):1379–1393, 2008.

Appendix A

User Manual

This is a practical and quick guide for a correct usage of the implemented system and functionalities already described.

- Prepare a 800x600 .png bitmap file of the environment *env.*
- Launch: *python create_graph_from_png.py.* It saves the .graphml file related to *env.*
- Launch: *python create_ptexplore.py.* Allows to draw the points which robots might visit. Saves the file in *env.exp.*
- Launch: *parsingData.py.* After specifying in the script the number of robots and the range, it creates the *.dat* file.
  Example: *bwopen_5r_500.dat*
- Launch: *python algorithm datfile obj_fun >outputFile.* Computes a solution and save the results in the output file.
  Example:
  python HBSS5.py bwopen_5r_500.dat time >resultsT_hbss_bwopen_500_tau5_5r.txt
- For analyzing the results and launch experiments, gather different solution files and create a folder.
  Then, in that folder, launch: *python parsingRes.py.*

Tools

- *Python*: coding language used to implement the scripts.
- *Gurobi*: Python library computing TSPs.
- *Matplotlib*: Python library for the creation of plots and diagrams.