ASSIGNMENT ALGORITHMS FOR MULTI-ROBOT TASK ALLOCATION IN UNCERTAIN AND DYNAMIC ENVIRONMENTS

A Dissertation

by

CHANGJOO NAM

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2016

Major Subject: Computer Science

ABSTRACT

Multi-robot task allocation is a general approach to coordinate a team of robots to complete a set of tasks collectively. The classical works adopt relevant theories from other disciplines (e.g., operations research, economics), but oftentimes they are not adequately rich to deal with the properties from the robotics domain such as perception that is local and communication which is limited. This dissertation reports the efforts on relaxing the assumptions, making problems simpler and developing new methods considering the constraints or uncertainties in robot problems.

We aim to solve variants of classical multi-robot task allocation problems where the team of robots operates in dynamic and uncertain environments. In some of these problems, it is adequate to have a precise model of nondeterministic costs (e.g., time, distance) subject to change at run-time. In some other problems, probabilistic or stochastic approaches are adequate to incorporate uncertainties into the problem formulation. For these settings, we propose algorithms that model dynamics owing to robot interactions, new cost representations incorporating uncertainty, algorithms specialized for the representations, and policies for tasks arriving in an online manner.

First, we consider multi-robot task assignment problems where costs for performing tasks are interrelated, and the overall team objective need not be a standard sum-of-costs (or utilities) model, enabling straightforward treatment of the additional costs incurred by resource contention. In the model we introduce, a team may choose one of a set of shared resources to perform a task (e.g., several routes to reach a destination), and resource contention is modeled when multiple robots use the same resource. We propose efficient task assignment algorithms that model this contention with different forms of domain knowledge and compute an optimal assignment under such a model.

Second, we address the problem of finding the optimal assignment of tasks to a team of robots when the associated costs may vary, which arises when robots deal with uncertain situations. We propose a region-based cost representation incorporating the cost uncertainty and modeling interrelationships among costs. We detail how to compute a sensitivity analysis that characterizes how much costs may change before optimality is violated. Using this analysis, robots are able to avoid unnecessary re-assignment computations and reduce global communication when costs change.

Third, we consider multi-robot teams operating in probabilistic domains. We represent costs by distributions capturing the uncertainty in the environment. This representation also incorporates inter-robot couplings in planning the team's coordination. We do not have the assumption that costs are independent, which is frequently used in probabilistic models. We propose algorithms that help in understanding the effects of different characterizations of cost distributions such as mean and Conditional Value-at-Risk (CVaR), in which the latter assesses the risk of the outcomes from distributions.

Last, we study multi-robot task allocation in a setting where tasks are revealed sequentially and where it is possible to execute bundles of tasks. Particularly, we are interested in tasks that have synergies so that the greater the number of tasks executed together, the larger the potential performance gain. We provide an analysis of bundling, giving an understanding of the important bundle size parameter. Based on the qualitative basis, we propose multiple simple bundling policies that determine how many tasks the robots bundle for a batched planning and execution.

ACKNOWLEDGEMENTS

The last five years at Texas A&M University were filled with great memories. Studying in a doctoral program, especially out of my homeland, was a big adventure. Although there have been many ups and downs, I am grateful to every moment that brought me to this place. The cover of dissertation shows that I am the sole author, but I would not be able to complete this without the unstinting supports from many teachers, colleagues, friends, and family. I would like to express my sincere gratitude to them.

I would like to express my deepest gratitude to my advisor, Dr. Dylan A. Shell. Since our first meeting on a sizzling summer day, he has been an enthusiastic advisor and also a thoughtful mentor. He guided me with his keen insight and exceptional intuition. He did not just try to teach me but threw great questions, which helped me to find the answers by myself. He encouraged me whenever I struggled with my research. Without his support, I would not be where I am today.

I am also very grateful for the help and advice from my committee members, Dr. Sergiy Butenko, Dr. Yoonsuck Choe, and Dr. Dezhen Song. Their insightful comments and constructive criticisms on my research made me to follow their high research standard.

The current and previous members of Distributed AI Robotics Lab have been great friends and also excellent fellow researchers. I would like to thank Jung-Hwan, Young-Ho, Sasin, Lantao, Plamen, Reza T., Taahir, Ben, Shawn, Yong, Reza O., Yulin, Eric, Kai, and Rui for the discussions, encouragements, and fun times.

I am also thankful to the Department of Computer Science and Engineering and its amazing staff. All the administrative work would not be that smooth without the support from the department. The free coffee in the lounge has played a big role for my productivity. The custodial staff has made the building a pleasant place to work in. More importantly, I appreciate the teaching assistantship opportunities from the department, which gave me valuable experiences in teaching. I am very grateful to Dr. Hyunyoung Lee, who supervised my TA work for several semesters. Her great management skills for the large-sized classes enabled me to carry out my duties without being overloaded.

I thank the colleagues at Amazon Robotics, who supported me to finish my internship successfully. I am proud of myself that I had worked with those brilliant people. I have so many fond memories of the summer in Boston.

Many friends have helped me stay happy through those difficult years. I will cherish all the unforgettable moments that we had.

I would not be able to complete my education without the unbounded love and endless support from my families in Korea. I deeply appreciate my parents' dedication and patience. Also, I am thankful to my brother for his efforts on family affairs during my absence. I would like to express my gratitude to my in-laws for having believed and supported me.

Most importantly, my wife Eunji has been my ideal companion through the last seven years. The life would not be always happy, but let's stand by each other whenever and wherever.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Multi-robot systems have advantages over single robots. The advances in computational power, sensor and communication technologies, and low-cost robot productions accelerate the advent of new technologies in various areas such as environmental monitoring [60], search and rescue [81], warehouse automation [106], surveillance [1], etc. Those applications make use of coordinated interaction among robots so they exhibit better performance than employing multiple individual robots in most cases. For a fluid and effective coordination, a team of robots needs a plan for task execution.

Conventionally, multi-robot task planning is decomposed into three steps as shown in Fig. 1.1. The task assessment step decomposes or combines tasks according to task specification and robot capabilities. Also, this step estimates the state of robots, tasks, and environments and evaluates the plans for executions. These estimates and evaluations are abstracted into a measure, such as cost or utility of task performance. The measure is an input to task allocation, which is a high-level decision of assigning a team of robots to a set of tasks. The decision is given to the next step, which generates motions of robots to perform the assigned tasks. Among these steps in task planning, we are interested in task allocation because of its generality and efficiency resulting from the effective abstraction.

Multi-robot task allocation (MRTA) addresses optimization of collective performance

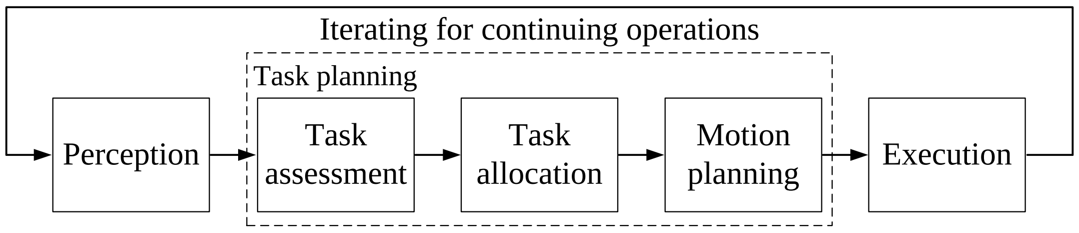
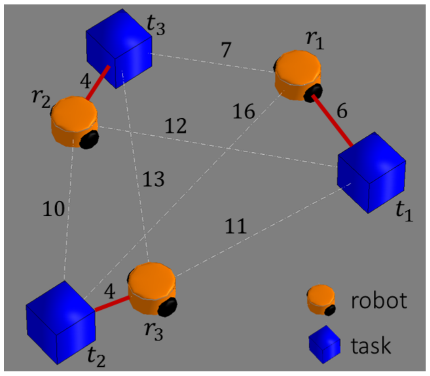
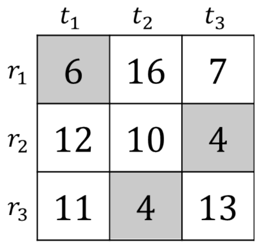


Figure 1.1: A classical model of task planning.



(a) An illustration of an MRTA problem.



	t_1	t_2	t_3
r_1	6	16	7
r_2	12	10	4
r_3	11	4	13

(b) The cost matrix constructed from (a).

Figure 1.2: An example of the ST-SR MRTA problem with a priori known costs. (a) Robots are assigned to tasks with the minimum sum of costs (solid lines). (b) Cost is a quantification of effort to complete a task. The shaded elements in the cost matrix represent an optimal assignment with the minimum sum of costs.

by reasoning about which robots in a team should perform which tasks. It aims to achieve a common goal through coordinated behavior. MRTA problems can be classified by robot capabilities, task requirements, and the length of planning horizon (instantaneous or extended periods of time) [43]. Among several MRTA problems, we are mainly interested in the problems in which each robot can perform only one task at a time and each task requires only one robot to execute it (falling into the ST-SR[1] class [43]). We consider instantaneous computations of task allocations using currently available information (Fig. 1.2a shows an example of the MRTA problem). However, the information may project future (or possible) states, so robots could have some foresight to deal with contingencies. Later, we extend this single step assignment computation to a longer time horizon as we consider realizations of the uncertainties.

1.1 Motivation

State estimates of robots working in dynamic and uncertain environments are prone to be outdated or inaccurate. The distributed nature of multi-robot systems worsens the es-

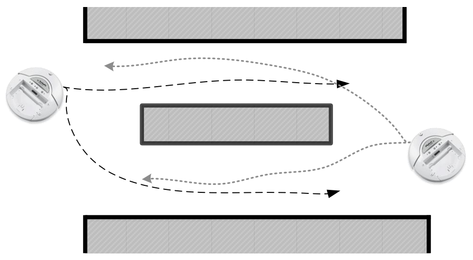
[1]**S**ingle-robot **T**ask and **S**ingle-task **R**obot.

timation because those systems are subject to additional constraints such as local perceptions and limited communication. Even though multi-robot systems have these properties, the vast majority of MRTA solution methods have been constructed by adapting theories from relevant disciplines including operations research, economics, and mathematics [43], which can be improved by domain properties.

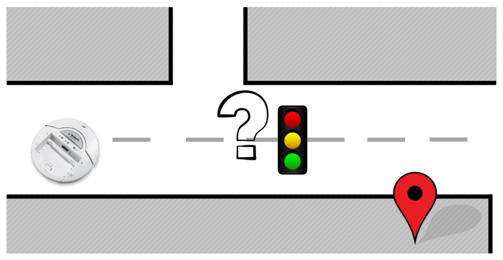
Task cost is an estimate of a value that shall be paid to perform tasks. In general, each robot estimates the costs, then these estimates are shared by robots over a communication network (Fig. 1.2b shows a cost matrix, which is a collection of costs of all robots). An optimal assignment is computed in a centralized manner (e.g., most approaches using the Hungarian method [65], an auctioneer [34], or a linear programming framework) or a decentralized manner (e.g., using a message passing mechanism [74]). The estimates could be inexact if the robot and task states change dynamically or the costs are computed based on state estimates which themselves change. In some cases, only probabilistic or stochastic information of costs is available to account for uncertainties in environments and state estimates.

We begin with investigating dynamically changing costs resulting from robot interactions. Robot interactions can be thought of as a run-time behavior so they are often treated in an online fashion. Often, the effect of interactions are treated as negligible. However, the optimal assignment under the assumption of negligible robot interactions may produce sub-optimal behaviors. Treating robot interactions by ad hoc methods (e.g., impromptu avoidance of other robots) may also worsen the quality of the assignment. Practically, robot interactions have considerable influence on costs, and the influence could be critical when robots contend for shared resources (Fig. 1.3a shows robot interactions over the constrained physical space).

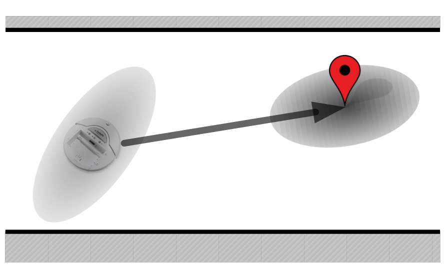
Uncertain costs arise when robots may face nondeterministic situations. While robots are executing their assigned task, the assumptions under which costs were calculated may



(a) Robot interactions.



(b) A nondeterministic situation.



(c) Uncertain state estimates.

(d) Uncertain task information.

Figure 1.3: Examples of dynamic or uncertain situations. (a) Robots have interactions over the shared resource (i.e., physical space). The interactions have considerable influence to costs. (b) A robot faces a nondeterministic situation, in which the traffic signal causes the uncertainty in the cost. (c) A robot has uncertain state estimates, which are represented by probability distributions. (d) A robot has limited information about future tasks.

turn out to be invalid; the environment may change, robots may have interactions or malfunctions, or a variety of other unexpected situations may emerge (Fig. 1.3b shows a nondeterministic situation—the traffic signal—that may change the cost). One solution which ensures a fluid response to these contingencies is to periodically re-calculate costs and recompute the optimal task assignments. However, this solution incurs computational and communication expense proportional to the desired recency.

As a consequence of considering uncertain costs, we also think about a model of cost uncertainty that is represented as probability distributions (an example is shown in Fig. 1.3c). If we use the distribution functions for costs, characterizing the distributions by some statistics (e.g., expected values) makes the problem tractable. Otherwise, we need to compute the sum of distributions, which is computationally expensive (e.g,. need convolutions). However, useful information about costs would be lost from the characterization. For example, the expected value cannot describe the uncertainty (or variability) modeled in the distribution. There is a need of more measures that characterize distributions sufficiently without losing important information.

Thus far, we have one common assumption that the set of tasks is finite and known a priori. In practical applications, only partial information of the task set might be accessible or the set is not even finite (Fig. 1.3d shows an example where tasks are not known before they arrive). One solution for the problems with sequentially arriving *online tasks*, where the locations and arrivals of tasks are stochastic, would be iterating conventional optimization methods for a sequence of the small subsets of tasks. Then the research problem reduces to a sequence of optimization problems for the finite subsets of tasks, which answers the question of *how to allocate* tasks to robots. But determining the size of each set precedes this optimization because including more tasks in the set uses a more informed view, potentially improving the optimization result.

1.2 Research Objective

In this dissertation, we aim to solve richer MRTA problems where some commonly made simplifying assumptions are relaxed. (i) We consider robot interactions as a non-negligible factor in task assignment optimization. We investigate how robot interactions over limited resources change costs, by modeling the consequence of the interactions precisely. (ii) We incorporate uncertainty into the region-based representation of costs where the region reflects possible contingencies during the execution of tasks. (iii) We consider a probabilistic formulation where costs are described by distributions. The formulation does not assume independent costs so inter-robot couplings are captured. (iv) We consider the setting where tasks are revealed sequentially, and it is possible to execute bundles of tasks.

1.3 Dissertation Contribution

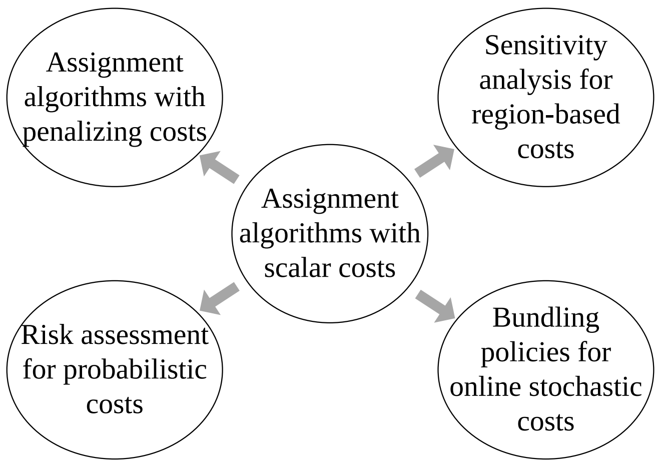

Assignment algorithms with penalizing costs
Sensitivity analysis for region-based costs
Assignment algorithms with scalar costs
Risk assessment for probabilistic costs
Bundling policies for online stochastic costs

Figure 1.4: An overview of the contributions of this dissertation. Our work attempts to expand the classical MRTA models in various directions.

Fig. 1.4 shows an overview of the contributions of this dissertation. This work expands the classical MRTA models in various directions to incorporate uncertainties and dynamics.

First, we propose assignment algorithms that include a model of robot interactions over shared resources. We introduce a generalization that allows for the additional cost incurred by resource contention to be treated in a straightforward manner. In this variant, robots may choose one of the shared resources (e.g., constrained space) to perform a task, and interference may be modeled as occurring when multiple robots use the same resource. By evaluating the consequence of robot interactions precisely through the model, an assignment computation can achieve global optimality instead of altering with the assignment during execution to deal with dynamically changing costs.

Second, we propose a region-based characterization of costs and algorithms that rea-

son about the optimality of the assignment within the region. The cost representation allows modeling contingencies in dynamic and uncertain environments. We develop an algorithm that employs the sensitivity analysis of linear programming so all that the possible optimal assignments within the cost region can be computed. The analysis enables the robots to avoid unnecessary re-assignment computations and reduce global communication when costs change during executions.

Third, we propose a probabilistic model with a risk preference specification and algorithms that analyze the sensitivity of the optimal assignment with respect to the preference. We use random variables for costs where the interrelationships among costs are not neglected, so inter-robot couplings can be incorporated into the formulation. We parameterize the problem with a risk preference that determines the importance between the mean and a risk measure on distributions. We analyze the sensitivity of assignment optima to particular risk valuations, which help in understanding the effects of risk on the problem.

Fourth, we propose task bundling policies where tasks are revealed sequentially over time. We consider synergistic tasks where planning with more tasks improves the performance, so the problem is not only in allocating tasks to robots, but also in determining the number of tasks that the robots plan and execute together. We identify two objectives that describe the performance of a team serving tasks over an infinite length (or a very long) horizon. We analyze the effect of the bundle size on performance. Based on the findings, we develop task bundling policies working with stochastic arrivals and probabilistic spatial distributions of tasks.

1.4 Organization

In Chapter 2, we provide related work and preliminaries. We describe a mathematical formulation of MRTA and sensitivity analysis of optimal assignments. Chapter 3 considers optimization of multi-robot task allocation when the overall performance of the team need not be a standard sum-of-cost model, but additional costs are incurred by robot interactions on shared resources. Chapter 4 and 5 consider the widely used optimal assignment problem formulation for task allocation but with two (a region-based and a probabilistic) representations that feature uncertain and interrelated costs. In Chapter 4, we propose algorithms using sensitivity analysis of linear programming to reduce dependencies on the centralized structure of multi-robot systems. In Chapter 5, we cast an MRTA problem as a scalarized biobjective assignment problem which minimizes the mean and a risk measure of a cost sum distribution. We develop algorithms for sensitivity analysis of the assignment subject to the scalarization parameter, which we term a risk preference. Chapter 6 tackles online task allocation problem with synergistic tasks. We propose model-based and model-free policies for task bundling and compare them with a baseline method which does not bundle but perform tasks instantaneously upon arrivals. Chapter 7 concludes and describes future directions.

2. RELATED WORK AND PRELIMINARIES

In this chapter, we review multi-robot task allocation (MRTA) approaches in general. We also provide a mathematical formulation of MRTA and sensitivity analysis of linear programming problem.

2.1 Related Work

The classical work in multi-robot coordination concentrates on designing the overall system architecture (e.g., [32], [41], [90], [105]). The early work focuses on developing working systems and demonstrations of implementations. As the field of study matures, various and specific topics have been addressed such as multi-robot path planning [29], formation control [23], exploration [91], and localization [8]. Among the various topics, the coordinating mechanism for a team of robot to achieve a common goal has received substantial attention because of its universality in multi-robot systems.

Gerkey and Matarić [43] propose a taxonomy for MRTA problems, which has been used widely, to provide a formal analysis of MRTA approaches. They suggest three axes of categories that classify MRTA problems by robot capabilities, task requirements, and the length of time horizon. Specifically, the first axis distinguishes between robots that can perform only one task at a time (ST) and that can simultaneously perform multiple tasks (MT). The second axis distinguishes tasks that can be performed by one robot (SR) and that may require multiple robots (MR). The third axis is for the length of time horizon where one category is for instantaneous assignments (IA) and the other is for time-extended assignments (TA)[1].

This classification is fairly clear and concise to describe many MRTA problems. It is

[1] All these acronyms stand for **S**ingle-**T**ask robots, **M**ulti-**T**ask robots, **S**ingle-**R**obot task, **M**ulti-**R**obot task, **I**nstantaneous **A**ssignment, and **T**ime-extended **A**ssignment.

convenient to confine the range of discussions but may exclude some important aspects of more complex task allocation problems (e.g., dependencies among robots, tasks, or costs). In a bid to encompass more sophisticated problems, Korsah et al. [63] recently suggest a new taxonomy that has one more axis for interrelated utilities and constraints. The new axis describes different types of dependencies among costs by no dependencies (ND), in-schedule dependencies (ID), cross-schedule dependencies (XD), and complex dependencies (CD). This taxonomy is more comprehensive since most MRTA problems have some degree of interrelationships among robots and tasks.

Still, the taxonomy might be insufficient to characterize some MRTA problems positioned close to the border lines of the classification. For example, a single-task robot may happen to perform other tasks while doing its own work (e.g., unintentionally exploring other robots' assigned area of an unknown space while it is moving to its assigned area) as discussed in Section 3.5 of [83]. Oftentimes, this kind of indirect coordination is not taken into account in the classification so the problem is treated in the ST-SR category. We also found another example regarding time horizon that a problem considers an instantaneous assignment may hedge against future states (e.g., computing a single-step allocation considering contingencies) [84]. In this case, the methodology may belong to the IA category (e.g., an assignment algorithm) but the problem itself belongs to the TA category.

Even though those classifications are not sufficiently inclusive for all MRTA problems, it is worth trying to characterize our problem with the taxonomy. We generally considers the ST-SR MRTA problem. In this case, instantaneous assignments can be computed in polynomial time by optimal algorithms such as the Hungarian method [65] and other algorithms [10], [16]. Computing time-extended assignments is NP-hard and usually solved by greedy or iterative methods. Our problems ambiguously range over both of the IA and TA classes since we consider instantaneous assignments, but they are computed based on information about possible future states. We also extend the planning horizon to infinity where the future tasks are described stochastically. In addition, our problems consider interrelated costs so belongs to the XD class, where costs of performing atomic or compound tasks have interdependencies.

Another widely used classification criteria is the structure of the task allocation mechanism. A multi-robot system may choose a centralized approach, where one of robots collects information from other robots and makes a decision for its team. Centralized approaches are widely used because they are simple to implement and easy to achieve global optimality since global information is available. The Hungarian method is one of the commonly used algorithms in centralized teams. Other linear programming-based methods are also widely used [6], [15]. However, these approaches show limitations when communication is not perfect and free. The central decision maker may suffer from computational burden if a team is large. If a system adopts a decentralized approach, each robot is in charge of its decision. Decentralized approaches are robust against single-point failures and do not need global communication. Market-based methods [34] distribute resource to pursue a maximal benefit for the whole team. There are some other methods such as a distributed version of the Hungarian method [44] and a message-passing method [74]. These approaches may not be able to attain global optimality if some important information is not delivered to some of constituents.

2.2 Preliminaries

In this section, we provide a mathematical formulation of the MRTA problem. Then we introduce sensitivity analysis of an optimal assignment that provides a prescribed region of costs where changes within the region do not impair the optimality of the current assignment.

2.2.1 A mathematical formulation of MRTA

The ST-SR-IA MRTA problem can be posed as an Optimal Assignment Problem (OAP). For n robots and n tasks[2], we assume we are given costs $c_{ij} \in \mathbb{R}^{\geq 0}$ that represent the cost of the i^{th} robot R_i performing the j^{th} task T_j for $i, j = 1, \cdots, n$. The robots should be allocated to tasks with the minimum cost sum. Let x_{ij} be a binary variable that equals to 0 or 1, where $x_{ij} = 1$ indicates that the R_i performs T_j. Otherwise, $x_{ij} = 0$. Then a mathematical description of the MRTA problem is

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i, \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j, \tag{2.3}$$

$$0 \leq x_{ij} \leq 1 \qquad \forall \{i, j\}, \tag{2.4}$$

$$x_{ij} \in \mathbb{Z}^{+} \qquad \forall \{i, j\}. \tag{2.5}$$

We make use of a matrix representation $\mathbf{X}^{*}$ that are $n \times n$ matrices representing a cost matrix and an optimal assignment of the problem, respectively.

2.2.2 Sensitivity analysis of optimal assignments

Sensitivity analysis (SA) has been studied for several decades in Operations Research to assess the robustness of optima for an optimization problem to perturbations in the input specification [38], [104], [70]. Analysis of an optimal assignment must compute a region

[2]This is without loss of generality, since if the numbers of robots and tasks are not equal, dummy robots or tasks would be inserted to make them equal. The costs of dummies have very large numbers so that they can be naturally excluded from the optimal assignment.

where costs within that region preserve the current optimal assignment. However, SA has found limited applicability to multi-robot task allocation problems: the analysis assumes that the decision maker (its counterpart in multi-robot systems is the central computation unit) is able to access all information off-line and has control all over the constituents without communication constraints. The physically distributed nature of multi-robot systems and their limited communication and computational resources pose challenges to the direct application of classical SA.

The OAP can be relaxed to a linear programming problem (LP) by removing the integral constraint[3]. The LP formulation of MRTA may make use of Sensitivity Analysis (SA) of an optimal assignment to yield a safe region of costs where all costs within the region preserve the current optimality. We provide a brief interpretation of the analysis for the MRTA problems, based on a comprehensive study of Ward and Wendell [104].

An LP problem corresponding to an MRTA problem can have more than one feasible solution. For each feasible solution, the variables x_{ij} $(i, j = 1, \cdots, n)$ can be divided into basic variables and nonbasic variables[4]. Let k be an index of a feasible solution. For each k, critical region R_k, a set of costs where an MRTA problem has the same optimal assignment for any cost $c \in R_k$, is defined as

$$R_k = \{c \in \mathbb{R}^{(n^2)} : \mathbf{c}_{N_k} - \mathbf{c}_{J_k} \mathbf{B}_k^{-1} \mathbf{A}_{N_k} \geq 0\} \tag{2.6}$$

where J_k and N_k indicate basic and nonbasic variables of the kth feasible solution, respectively. $\mathbf{B}_k$ and $\mathbf{A}_{N_k}$ are constraint matrices of basic variables and nonbasic variables.[5] $\mathbf{c}_{J_k}$ and $\mathbf{c}_{N_k}$ are cost vectors of basic and nonbasic variables. Note that the critical region R_k

[3]This relaxation depends on $c_{ij} \in \mathbb{Q}$, which is exact for any implementation.

[4]A variable is basic if it corresponds to one of the vectors in the basis, given a feasible basis to a linear-programming problem.

[5]A constraint matrix of an optimization problem consists of coefficients of variables. $\mathbf{B}_k$ is a set of columns corresponding to basic variables. Similarly, $\mathbf{A}_{N_k}$ corresponds to the coefficients of nonbasic variables.

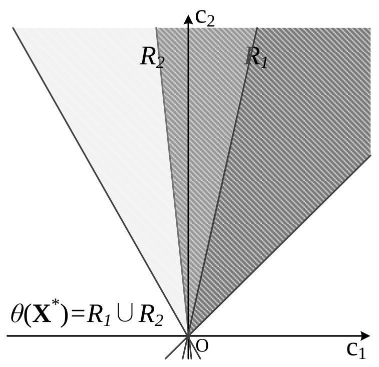


Figure 2.1: A 2-D example of $\theta(\mathbf{X}^*)$. Any cost in the set has the same optimal assignment.

is formed by linear boundaries with nonempty interiors.

However, there is an additional complexity because the MRTA problem is degenerate (see Appendix B). The critical region R_k is not a complete description of the region which preserves optimality of the current assignment. The complete set is

$$\theta(\mathbf{X}^*) = \bigcup_{k \in H} R_k, \tag{2.7}$$

where $H = \{k : \mathbf{X}^*_{J_k} = \mathbf{B}_k^{-1}, \mathbf{X}^*_{N_k} = 0\}$, which is the union of critical regions of all degenerate solutions. Note that $\theta(\mathbf{X}^*)$ is also a polyhedral set [104, Theorem 17] consisting of linear boundaries that cross the origin (there might be overlaps among R_k). Fig. 2.1 shows a 2-D example of $\theta(\mathbf{X}^*)$. Notice that the smallest nontrivial MRTA problem has 4 dimensions (2 robots and 2 tasks), so we show this trivial case for a better visualization.

An $n \times n$ MRTA problem has $2n-1$ basic variables and $(n-1)^2$ nonbasic variables. To compute (2.7), we must identify the basic and nonbasic variables of the k^{th} feasible solution. The n variables corresponding to costs in the optimal assignment are basic variables, but the degeneracy means that the remaining $n-1$ basic variables cannot be identified directly. Thus, we shall choose the $n-1$ basic variables from the remaining $n^2 - n$ variables to complete a feasible solution, yielding a total of $\binom{n^2-n}{n-1}$ feasible solutions. Note, however, that despite the set being large the interiors of R_k may overlap so $\theta(\mathbf{X}^*)$ can be covered by a small subset of R_k.

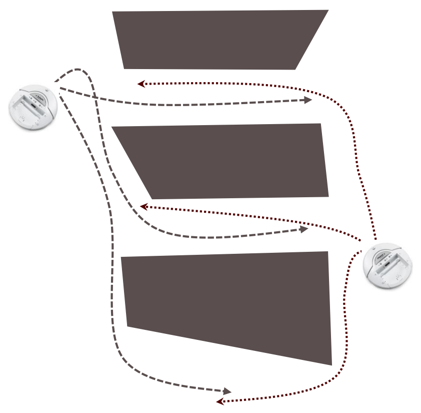
3. ASSIGNMENT ALGORITHMS FOR MODELING RESOURCE CONTENTION IN MULTI-ROBOT TASK ALLOCATION

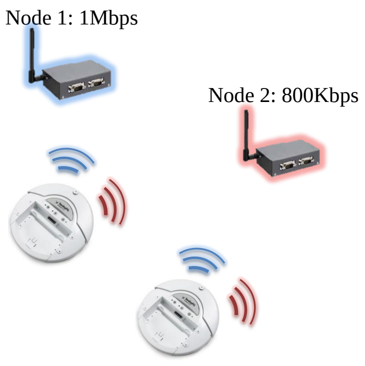
MRTA addresses optimization of collective performance by reasoning about which robots in a team should perform which tasks. Even starting with the classical work, many different approaches have been proposed, such as behavior-based [90], [105] and market-based [17], [32], [42] task allocation. Although resource contention and physical interference have long been known to limit performance [46], [45], [97], the vast majority of MRTA work considers settings for which interference is treated as negligible (*cf.* review in [43]). This limits the applicability of these methods and computing a task assignment under assumptions of noninterference may produce suboptimal behavior even if the algorithm solves the assignment problem optimally. Several authors have proposed task allocation approaches that model or avoid interference (usually physical interference), see for example, [28], [51], [25], [92] (a summary is shown in Table 3.1.). These works, however, do not set out to achieve global optimality, or understand the computational consequences of a model of interference.

In this chapter, following the lead of early and practical work, we assume a networked

Table 3.1: A summary of algorithms that consider interference among robots.

Authors & paper	Way to deal with interference	Poly-time?	Optimal?
Dahl et al. [28]	Use reinforcement learning to distribute resources to robots	No	No
Guerrero and Oliver [51]	Include the effect of interference in the utility function of the auction method	No (deadline)	No
Choi et al. [25]	Use a market-based distributed agreement protocol that guarantees a conflict-free assignment	Yes	No (provably good solution)
Pini et al. [92]	Spatially partition tasks to reduce interference	No	No


(a) Physical space



(b) Communication bandwidth

Figure 3.1: Two examples of resources with limited capacities that must be shared in most practical contexts. Both communication and space contention cause performance to scale sub-linearly with the number of robots.

system in which all information is known by at least one robot that is responsible for optimizing task allocation. In practice, this robot can be dynamically elected robot from amongst the team. We also assume the ST-SR-IA MRTA problem, which can be posed as an Optimal Assignment Problem (OAP). The OAP is well-studied and can be cast as an integral linear program which is in complexity class P. This conventional MRTA problem does not specify how robots use resources so it is unable for it to account for interference incurred by sharing resources. Instead, it assumes that resources are individually allocated to robots or, if shared, that they impose no limits.

In our problem, however, robots may have to choose between resources used to perform tasks (e.g., several routes to reach a destination), as shown in Fig. 3.1, and the costs of performing the tasks may vary depending on the choice. If several robots use the same resource (reflected in a relationship between their choices), we allow interference between them to be modeled. Inter-agent interference (as described in Fig. 3.2) is treated mathematically as a penalization to the cost of performing that task. In this manner, we can model

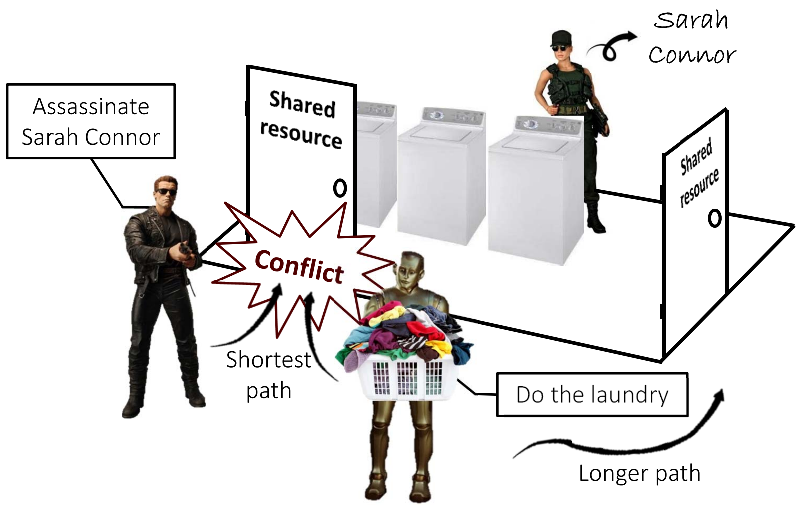


Figure 3.2: A specific example of resource contention: two robots choose the shortest path to perform their tasks, they should compute their paths to avoid interaction with each other. When the right robot chooses the longer path via the door on the far right, the sum of distances is larger, but it minimizes cost when resource contention is considered.

shared resources and generalize the conventional MRTA problem formulation to include resource contention. The result is an optimization problem for finding the minimum-cost solution including the interference induced penalization cost. We term this the multi-choice assignment problem with penalization (mAPwP). The model we introduce allows a robot to make a selection from among multiple means by which it could perform a task. Naturally, the penalization depends on the particular selection.

In general, there are many ways penalization costs could be estimated. When evaluation of the interference is polynomial-time computable, we call this the mAPwP problem with polynomial-time computable penalization function (P-type mAPwP). Even with a cheaply computable penalization function, we show that the P-type problem is NP-hard[1]. We also investigate two other problems that have particular forms of penalization functions: linear and general convex penalization functions. We show that the two problems

[1]Adding the notion of multiple choices does not change the complexity class, which is P. However, introducing the penalization function makes the problem hard.

are in P and NP-hard, respectively. We provide an exact algorithm and two polynomial-time algorithms for the problems. The algorithms are domain-independent so that it can be used for many multi-agent scenarios that have quantifiable interference between agents.

The remainder of this chapter is organized as follows. Section 3.1 discusses the related literature on optimization methods for MRTA. Section 3.2 defines the problem mathematically, and Section 3.3 describes the NP-hardness results. Section 3.4 presents algorithms, and Section 3.5 extends the suggested modeling method of resource contention to another interrelated costs. Section 3.6 describes experiments, and the final section concludes.

3.1 Related Work

Recent studies dealing with limited shared resources in multi-robot systems are mainly focused on the multi-robot path planning (MPP) problem: Alonso-Mora et al. [2] employ a mixed-integer quadratic programming methods to optimize trajectories of robots while avoiding collisions; Yu and LaValle [107] propose an integer linear programming method to find collision-free paths for multiple robots; He and van den Berg [52] suggest an MPP algorithm that consists of macro-, micro-, and meso-scale planners. Their meso-scale planner considers groups of other robots as a coherent moving obstacle while the micro-scale planner locally avoids individual obstacles. Those methods quickly find high-quality solutions. However, their approaches are domain-specific so not appropriate for general problems where robots contend for arbitrary shared resources, not necessarily only physical space. Moreover, resources modeled in [107] are able to accommodate only one robot at each time step, which is restrictive to model real-world applications. In [52], the micro-scale collision avoidance is based on local observations, and it does not achieve global optimality. We are not aware of previous hardness results with respect to resource sharing in multi-robot systems.

The equivalence of the classical assignment problem by a network flow problem has been well known for decades. This may lead to the suggestion that one can prevent interference by imposing additional constraints in the form of capacity constraints in the flow formulation. This can be solved by a centralized manner [35] or a distributed manner [66], [77]. However, that approach models interference as a binary penalization, which is zero or infinite, whereas incurred by resource contention are more widely applicable if the interference is modeled as a continuous function that increases proportionally to the amount of interference. (See, for example, our use of published and validated traffic models in Section 3.6.)

The approach of imposing constraints to restrict robots from using shared resources is used in many MRTA algorithms such as [101], [96], [99], [110], [24], [56]. This approach is widely used because of its simplicity since the constraints can be constructed once restrictions on resources are identified (e.g., the maximum number of robots using a shared resource). However, such constraints satisfy some models of shared resource, but the models are not adequately rich to describe the problem precisely. For example, if a capacity constraint is imposed for a shared resource, an allocation that violates the constraint cannot be considered at all. However, a shared resource can be used without a capacity limit but with some additional costs as more robots use the resource. Inversely, there could be additional costs even though the number of robots using a shared resource is less than a capacity. In addition, [73], [74] also consider MRTA problems where tasks have dependencies. The inter-task dependencies are caused by precedence or deadline of tasks. The dependencies are handled by imposing constraints. Again, this approach may not be describes some problems precisely. For example, a shipping task could miss its deadline if a penalty is paid for not fulfilling the due date.

An alternative is for the P-type problem can be cast as a linearly constrained 0-1 programming problem, with the penalization function incorporated into the objective function with the cost sum. The objective function is optimized over a polytope defined by the mutual exclusion and integral constraints. The results in this chapter suggest that one can have an optimal solution in polynomial time if the penalization function is linear. When the penalization is more complex, a common method to solve the problem is enumeration, for example using the branch-and-bound method, but its time complexity in the worst case is as bad as that of an exhaustive search; rather more insight is gained by employing the method we introduce in this chapter. Many practical algorithms [68, 58, 27] are suggested in the literature, but they also have exponential running time in the worst case. Linearizing the complex penalization function could be an alternative to have polynomial running time but has no performance guarantee.

Lastly, Roughgarden [95] introduces noncooperative routing games in which each agent chooses a complete route between a source and a sink in a network in congestion-sensitive manner. Routing games have the objective of minimizing the sum of traffic costs including additional costs from congestion, which is same with the multi-vehicle traffic problem used in the experiments (Section 3.6.3). It is interesting that selfish agents are able to find an optimal set of routes. However, routing games confine their applications to routing problems on physical resources (e.g., roads) so they are limited to deal with general resource contention.

3.2 Problem Formulation

3.2.1 Bipartite multigraph

The mAPwP problem can be expressed as a bipartite multigraph. Let $G = (R, T, E)$ be a bipartite multigraph consisting of two independent sets of vertices R and T, where $|R| = n$ and $|T| = m$, and a collection of edges E. An edge is a set of two distinct vertices denoted (i, j) and incident to i and j. Each edge in G is incident to both a vertex in R and a vertex in T, and p_{ij} is the number of edges between two vertices. The vertices in R and T can be interpreted as n robots and m tasks, respectively. An edge is a way in which a robot

Table 3.2: Nomenclature.

$G(R, T, E)$	a bipartite multigraph consisting of two disjoining sets R and T and a collection of edges E;
L	the bit length of input variables of an instance;
N_Π	the number of all assignments;
$Q(\cdot)$	the penalization function;
Q_l	the penalization function of the l-th resource;
Q^s	the penalization of s-th assignment;
X^*	the optimal assignment;
$X^{s-/+}$	the s-th assignment before/after penalization;
c_{ijk}	the cost of performing the j-th task by the i-th robot in the k-th manner;
c^*	the cost sum of the optimal assignment;
$c^{s-/+}$	the cost sum of the s-th assignment before/after penalization;
d	the length of a road;
i	the index of vertices in R (robots);
j	the index of vertices in T (tasks);
k	the index of edges in E (choice);
n	the number of vertices in R (robots);
n_l	the number of robots on the l-th resource;
m	the number of vertices in T (tasks);
p_{ij}	the number of choices between $r_i \in R$ and $t_j \in T$;
x_{ijk}	the binary variable that indicates that the i-th robot performs the j-th task in the k-th manner;
s	the s-th best assignment in terms of optimality;
v_f	the traffic flow speed of a road;
β	the coefficients of a penalization function;
η	the ratio of an approximated solution to an optimal solution ($\eta = c'^*/c^*$);
λ	the slope of the headway-speed curve;
ρ	the traffic density of a road;
ρ_j	the jam density of a traffic road;

may use resources, for which it expected to select one among p_{ij} choices for a given task. The precise interaction between resources is modeled via penalization function, described next.

3.2.2 *Multi-choice assignment problem with penalization (mAPwP)*

Given n robots and m tasks, the robots should be allocated to tasks with the minimum cost. Each allocation of a robot to a task can be done via one of the p_{ij} choices where i and j are indices of the robots and the tasks, respectively. Each of the p_{ij} choices represents some set of resources used by a robot to achieve a task. The multiple choices indicate the resources can be used in many ways. We assume we are given c_{ijk}, the interference-free cost of the i-th robot performing the j-th task through the k-th choice. Let x_{ijk} be a binary variable that equals to 0 or 1, where $x_{ijk} = 1$ indicates that the i-th robot performs the j-th task in the k-th manner. Otherwise, $x_{ijk} = 0$.

In problem domains where multiple robots share resources, use of the same limited resource will typically incur a cost. We model this via a function which corrects the interference-free assignment cost (i.e., the linear sum of costs) by including the additional cost of the effects of resource contention ($Q(\cdot)$ in (3.1))[2]. We assume that the cost and the penalization are nonnegative real numbers. We also permit the cost to positive infinity when interference is catastrophic (or, for example, only one robot is permitted to use the resource). We assume $n = m$. If $n \neq m$, dummy robots or tasks would be inserted to make $n = m$. Then a mathematical description of the mAPwP problem is

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{p_{ij}} x_{ijk} c_{ijk} + Q(x_{111}, x_{112}, \ldots, x_{11p_{11}}, \ldots, x_{nmp_{nm}}), \tag{3.1}$$

[2]The formal definition of $Q(\cdot)$ will be shown in Section 3.2.3.

subject to

$$\sum_{j=1}^{m}\sum_{k=1}^{p_{ij}} x_{ijk} = 1 \qquad \forall i, \tag{3.2}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{p_{ij}} x_{ijk} = 1 \qquad \forall j, \tag{3.3}$$

$$0 \leq x_{ijk} \leq 1 \qquad \forall \{i,j,k\}, \tag{3.4}$$

$$x_{ijk} \in \mathbb{Z}^{+} \qquad \forall \{i,j,k\}. \tag{3.5}$$

We note that (3.5) is superfluous if no penalization function is considered or $Q(\cdot)$ is linear, because the constraint matrix satisfies the property of totally unimodular (TU) matrix. Specifically, an optimization problem with a linear objective function has only integer solutions if its constraint matrix satisfies totally unimodularity [30], so the integral constraint is not necessary.[3]

3.2.3 Penalization

The penalization function maps a particular assignment to the additional cost associated with the interference. In the formulation of mAPwP earlier, $Q(\cdot)$ denotes the penalization function in most general terms. If the mAPwP is with a polynomial-time computable $Q(\cdot)$, it is the P-type problem. The input domain for Q has $\sim O(\max\{n,m\}! \cdot (\max\{p_{ij}\})^{\min\{n,m\}})$ elements; in most cases a penalization function is more conveniently written in some factorized form. One example is if one is concerned only with the number of robots using a resource, not precisely the identities of the robots that are. If $Q_l(n_l)$ is the penalization function of the l-th choice where n_l is the number of robots for that choice,

[3]The standard treatment of the Optimal Assignment problem without a penalization factor for task allocation (e.g., in [43]) considers only a bipartite graph (i.e., $\forall_i \forall_j p_{ij} = 1$). Although TU is well-known for the problem, we believe this to be the first recognition of this fact for the problem above.

Table 3.3: A summary of the mAP problems.

Problem	Description
mAPwP	The multi-choice assignment problem with penalization
P-type	The mAPwP problem with any penalization functions that are polynomial-time computable
DP-type	The decision version of the P-type problem
C-type	The mAPwP problem with convex penalization functions
L-type	The mAPwP problem with linear penalization functions

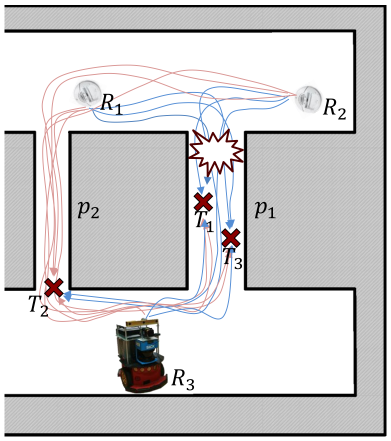
then the total penalization could be written as:

$$\begin{aligned} &Q(x_{111}, x_{112}, \ldots, x_{11p_{11}}, \ldots, x_{nmp_{nm}}) \\ &= Q_1(n_1) + Q_2(n_2) + \ldots + Q_q(n_q) \\ &= \sum_{l=1}^{q} Q_l(n_l). \end{aligned} \tag{3.6}$$

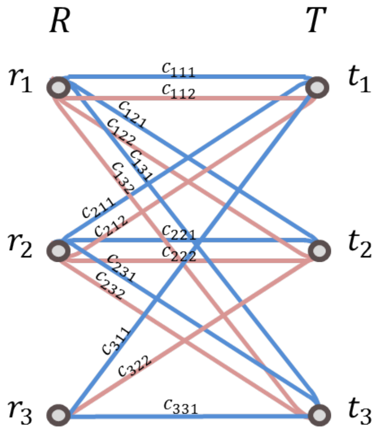
where q is the total number of choices in an environment. If the robots are homogeneous, n_l is the same as the number of robots on the l-th choice. Otherwise, each robot has a weight that represents the occupancy of the robot. The P-type problem is a general problem that $Q(\cdot)$ can be any form of function. If $Q(\cdot)$ is convex, the mAP becomes the mAP with convex penalization function (C-type mAPwP). Especially, it comes to be the mAP with linear penalization function (L-type mAPwP) if $Q(\cdot)$ is linear. The descriptions of the problems are summarized in Table 3.3.

3.2.4 Examples

An example of the mAPwP is shown in Fig. 3.3a. The goal is to minimize the total traveling time by distributing robots (R_1, R_2 and R_3) to three destinations (T_1, T_2 and T_3). R_1 and R_2 can use all the paths, but R_3 cannot use the passage p_2 because R_3 is wider than the passage. A weighted bipartite multigraph that is equivalent to the example is shown in Fig. 3.3b. The graph has $|R| = |T| = 3$ vertices, and every pair of vertices has 2 edges except for $p_{31} = p_{32} = p_{33} = 1$. There will be interference, for example, if both R_1 and



(a) An example of the mAPwP.



(b) The equivalent graph representation.

Figure 3.3: An example of the mAPwP and its graph representation. (a) Robots have a choice between routes to reach their destinations, but interference will occur if a passageway is shared (e.g., if both R_1 and R_2 try to reach destinations via p_1.) (b) A weighted bipartite multigraph representation for this example. An edge between r_i and t_j represents the use of a resource to perform the j-th task by the i-th robot, and its weight (c_{ijk}) is a cost associated with performing the task by the robot. x_{ijk} is a binary variable that indicates allocation of a robot to a task through a resource (the variables are omitted for clarity).

R_2 try to reach destinations on p_1, so a time delay is incurred which must be added to the total traveling time.

Types of shared resources need not be limited to physical space. A family of cooperative information collecting missions could have resource contention on shared communication channels. The mission is collecting information, such as pictures, depth information, or audio source, from environments and transmitting them to a central repository while minimizing the sum of completion time. Each robot is required to choose one of the locations in an environment and transmit collected data through one of several private wireless networks that have different bandwidth.[4] Data transmission time depends on the size of the chosen channel's throughput and the data size, but additional transmission time

[4]To simplify the problem, we assume that the time for approaching to a location and transmitting data dominates the time for other tasks such as data acquisition. We also assume that physical space is enough to perform tasks without interference among robots.

occurs if the traffic exceeds the bandwidth. This example of network congestion can be formulated similarly with the physical space case.

3.3 NP-Hardness of mAPwP Problems

In this section, we show the P-type and C-type problems are NP-hard optimization problems, and the L-type problem is in P. We prove the corresponding decision version of the P-type (DP-type) is NP-complete to prove the P-type problem is an NP-hard optimization problem [57]. Then we briefly describe the L-type problem is in P and show the C-type problem is NP-hard.

3.3.1 The P-type problem is NP-hard

Theorem 3.1 The DP-type problem is in NP.

Proof. The DP-type problem simply asks whether an assignment has cost less than a given threshold.

Input: n robots, m tasks, p_{ij} choices, a polynomial-time computable penalization function Q, and costs of edges c_{ijk}, a constant α.

Question: Is the penalized cost of a given assignment less than α?

Certificate: An arbitrary assignment x_{ijk}.

Algorithm:

```
1 Check whether the assignment violates any constraints
2 Calculate the total cost of the assignment
3 Penalize the cost by the penalization function
4 Check whether the penalized cost is less than $\alpha$
```

This is polynomial-time checkable so that the DP-type problem is in NP. □

Theorem 3.2 The DP-type problem is NP-hard.

Claim. The proof is based on relation to the classic boolean satisfiability problem. The

3-CNF-SAT problem asks whether a given 3-CNF formula is satisfiable or not. It is a well-known NP-complete problem. If 3-CNF-SAT $\leq_{\mathrm{P}}$ DP-type, then the DP-type problem is NP-hard.

Proof. The reduction algorithm begins with an instance of 3-CNF-SAT. Let $\Phi = C_1 \wedge C_2 \wedge ... \wedge C_k$ be a 3-CNF boolean formula with k clauses over n variables, and each clause has exactly three distinct literals. We shall construct an instance of the DP-type problem where $p_{ij} = 1$ ($i = 1, ..., n$ and $j = 1, ..., 2n$) such that Φ is satisfiable if and only if the solution of the instance of DP-type problem has cost less than a constant α.

We construct a bipartite multigraph $G = (R, T, E)$ as follows. We place n nodes r_1, r_2, ..., $r_n \in R$ for n variables and $2n$ nodes $t_1, f_1, t_2, f_2, ..., t_n, f_n \in T$ for truth values (true and false) of the variables. For $i = 1, ..., n$ and $j = 1, ..., 2n$, we put edges $(r_i, t_i) \in E$ and $(r_i, f_i) \in E$ where t_i and $f_i \in T$. The costs of the edges are given by c_{ij}. In addition, we construct an assignment by assigning vertex i in R to vertex j in T only when $x_{ij} = 1$ for $i = 1, ..., n$ and $j = 1, ..., 2n$. (Note that $x_{ij} \in \{0, 1\}$.)

Now, we construct a function Φ_{J} as follows. Each clause in Φ is transformed to a sum of terms in parentheses so that the terms correspond to the three literals in the clause. For a positive literal, we put x_{ij} where i is equal to the index of the literal and $j = 2i-1$ whereas $j = 2i$ for a negative literal. Disjunctions of clauses are transformed to multiplications. A penalization of an assignment is defined as

$$Q = \begin{cases} 0 & \Phi_{\mathrm{J}} > 0 \\ \mathrm{N} & \text{otherwise}, \end{cases} \tag{3.7}$$

where N is a large number. If Φ_{J} has a solution which makes $\Phi_{\mathrm{J}} > 0$, the penalization is zero. Therefore, the cost of the assignment is $\sum_{i,j} c_{ij}x_{ij}$ and $Q = 0$ so the assignment has the total cost $\sum_{i,j} c_{ij}x_{ij}$. Otherwise, it will have a large nonzero penalization such as N.

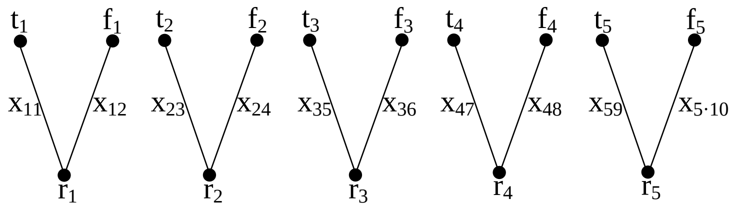


Figure 3.4: The DP-type problem derived from the 3-CNF formula $\Phi = (x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_3 \vee \neg x_1 \vee \neg x_2)$. A satisfying assignment of Φ has $x_1 = 1, x_2 = 1, x_3 = 1$, and x_4, x_5 either 0 or 1. Corresponding assignment is that $x_{11} = 1, x_{12} = 0, x_{23} = 1, x_{24} = 0, x_{35} = 1, x_{36} = 0$. The values of other elements do not affect the satisfiability of Φ. This assignment makes $\Phi_{\mathrm{J}} > 0$.

We can easily construct Q from Φ in polynomial time.

As an example, consider the construction if we have

$$\Phi = \ (x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_3 \vee \neg x_1 \vee \neg x_2), \tag{3.8}$$

then the transformation is shown in Fig. 3.4. Φ has five variables so five nodes and ten nodes are placed in R and T, respectively. The nodes in R and T which have the same subscripts are connected. We produce function:

$$\Phi_J = (x_{11} + x_{23} + x_{48}) \cdot (x_{23} + x_{47} + x_{5\cdot 10}) \cdot (x_{35} + x_{12} + x_{24}), \tag{3.9}$$

and its penalization will be 0 or N depending on the assignment.

We show that this transformation is a reduction in a little more detail. First, suppose that Φ has a satisfying assignment. Then each clause contains at least one literal that true is assigned, and each such literal corresponds to a matching of r_i and t_i. On the contrary, a literal assigned false corresponds to a matching of r_i and f_i. Thus, assigning truth values to the literals to make Φ satisfied yields matchings between R and T. We claim that the matchings are an assignment which makes $\Phi_{\mathrm{J}} > 0$. The assignment makes each sum of three terms (in parentheses) at least 1 so that Φ_{J}, a multiplication of the parenthesized terms, is greater than or equal to 1. Therefore, by the construction, we can get the total cost of the assignment and answer whether the cost is less than α.

Conversely, suppose that the DP-type problem has an assignment that makes $\Phi_{\mathrm{J}} > 0$. We can assign truth assignments to the literals corresponding to the matchings between R and T so that each clause has at least one variable which is true. Since each clause is satisfied, Φ is satisfied. Therefore, 3-CNF-SAT $\leq_{\mathrm{P}}$ DP-type.[5] $\square$

In the example of Fig. 3.4, a satisfying assignment of Φ has $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, and x_4, x_5 either 0 or 1. Corresponding matchings in DP-type are that r_1 and t_1, r_2 and t_2, r_3 and t_3 while r_4 is matched to either t_4 or f_4. Also, r_5 is matched to either t_5 or f_5. Therefore, the assignment is $x_{11} = 1, x_{12} = 0, x_{23} = 1, x_{24} = 0, x_{35} = 1, x_{36} = 0$. The values of other elements do not affect the satisfiability of Φ. This assignment makes $\Phi_{\mathrm{J}} > 0$.

Corollary 3.3 By Theorem 3.1 and 3.2, the DP-type problem is NP-complete. Therefore, the P-type problem is an NP-hard optimization problem.

3.3.2 The L-type problem is in P

Mathematically, the L-type problem can be cast as an integer linear programming problem whose constraint matrix satisfies the property of totally unimodularity. This problem can be solved in polynomial time as described in [30, Corollary 2.2]. Therefore, the L-type problem is in P.

3.3.3 The C-type problem is NP-hard

The mAP with a convex quadratic penalization function (CQ-type) is a proper subset of the C-type problem, and is a natural next step after examining L-type problem. The

[5] There can be a simplification of the reduction. We can construct a function Φ_{J} from any of the 3-CNF-SAT problem Φ. We define a polynomial-time penalization function Q as (3.7). Then solving the DP-type problem solves the corresponding instance of the 3-CNF-SAT problem.

CQ-type problem has the form

$$\min \{x^T Hx + cx \colon Ax \leq b, x \in \{0,1\}\} \tag{3.10}$$

where H is positive semidefinite and symmetric, c is nonnegative, A is TU, and b is integer.

The following binary quadratic programming (BQP) is an NP-hard problem [9, Theorem 4.1]. The BQP problem is

$$\min \{y^T My + dy \colon A'y \leq b', y \in \{0,1\}\} \tag{3.11}$$

where $M = L^T DL$, $D = I$, $d = 0$, L is TU and nonsingular, A' is TU, and b' is integer.

Theorem 3.4 The CQ-type problem is NP-hard.

Claim. If M is symmetric and positive semidefinite, we can reduce any BQP to an instance of the CQ-type problem. Namely, BQP $\leq_{\mathrm{P}}$ CQ-type.

Proof. Since $D = I$, $M = L^T L$. Then $(L^T L)^T = (L)^T (L^T)^T = (L^T L)$. Thus, M is symmetric.

For any column vector v, $v^T L^T Lv = (Lv)^T Lv = (Lv) \cdot (Lv) \geq 0$. Thus, $L^T L$ is positive semidefinite. Therefore, BQP $\leq_{\mathrm{P}}$ CQ-type as we claimed. $\square$

Lemma 3.5 CQ-type $\subsetneq$ C-type.

Corollary 3.6 By Theorem 3.4 and Lemma 3.5, the C-type problem is NP-hard.

3.3.4 A polynomial-time solvable class of the C-type problem

The C-type problem is a nonseparable convex optimization problem. If a C-type problem can be converted to a separable convex optimization problem without breaking the totally unimodularity of the constraint matrix, the problem is solvable in polynomial time [55] and Alg. 4.2 in [55] is an optimal algorithm for these cases. Note that a nonseparable problem can be transformed to a separable problem by substituting nonseparable dependent polynomials with additional independent variables and imposing additional constraints[6] (see [18, Table 13.1] and Appendix A for more detail and an example). Since the cost sum part of the objective function in (3.1) is separable in itself, only the penalization function is subject to conversion. A constraint matrix after the conversion is

$$A_{\mathrm{SP}} = \begin{bmatrix} A & 0 \\ A_{\mathrm{N}} & -I \end{bmatrix} \tag{3.12}$$

where A is the original TU constraint matrix, and $[A_{\mathrm{N}} - I]$ are newly imposed constraints. A is $n \times mp$, A_{N} is $w \times mp$, 0 is $n \times w$, and I is $w \times w$ matrix where w is the number of newly added variables. According to the properties of TU matrix, we have the following properties of A_{SP}:

- $[A\ 0]$ is TU.
- If A_{N} is TU, $[A_{\mathrm{N}}\ -I]$ is also TU.
- Joining two arbitrary TU matrices is not guaranteed to make a TU matrix. Thus, A_{SP} may not be TU although both of $[A\ 0]$ and $[A_{\mathrm{N}}\ -I]$ are TU.
- If A_{N} is not TU, then A_{SP} is not TU.

Since a TU A_{N} could make A_{SP} either TU or non–TU, the totally unimodularity of A_{SP} should be checked. The definition of TU (i.e., the determinant of every square submatrix has value -1, 0, or 1) or a necessary and sufficient condition described in [22] can be used to check the totally unimodularity. However, using those methods for the entire A_{SP} could be computationally expensive for large-sized problems. We suggest a preliminary test to see if a transformation breaks the totally unimodularity of the original problem.

[6]Theoretically, any optimization problem can be restated as a separable program, but this is of limited practically as the number of the additional variables and constraints is large [18].

Theorem 3.7 Separable convex integer optimization problem, whose constraint matrix is not TU, is NP-hard.

Proof. Separable integer linear programming (ILP) problem is a special case of the separable convex integer programming problem. An ILP, whose constraint matrix is not TU, is NP-hard [59] regardless of whether it is separable or not. Therefore, the separable convex integer optimization problem, whose constraint matrix is not TU, is NP-hard. □

Thus, a non–TU A_{N} makes the C-type problem NP-hard by Theorem 3.7. A preliminary test that is checking the totally unimodularity of A_{N} before checking the entire A_{SP} may save time, because A_{SP} needs not to be checked if A_{N} is not TU. However, A_{SP} shall be checked if A_{N} is TU since a non–TU A_{N} is a sufficient condition, but not a necessary condition, to make non–TU A_{SP}.

3.3.5 Remark on the hardness results

There is a significance in the NP-completeness (not merely the NP-hardness) result of the DP-type problem when the penalization function is polynomial-time computable. The problem with polynomial-time solvable penalization functions has a spectrum of the hardness from P to NP-complete; the upper bound is perhaps surprising. Many MRTA problems become NP-hard when richer and more precise descriptions (e.g., additional constraints) are added to the problem formulation [43]. While the problem is not expected to be polynomial-time solvable, even if some polynomial-time algorithms did solve all NP-complete problems, the NP-hard ones might remain. If a problem has a non-polynomial-time-solvable penalization function, the problem becomes NP-hard. The spectrum is a concise visualization of understanding how hard the problem is depending on the form of the penalization function.

3.4 Algorithms for mAP Problems

In this section, we devise algorithms for mAP problems. The exact algorithm for the P-type problem recursively enumerates unpenalized assignments and their costs from the best assignment in terms of optimality, by calling a combinatorial optimization algorithm for each iteration. However, no enumeration and optimization algorithm exists for multigraphs, so we must extend Murty's ranking algorithm [82] and the Hungarian method [65] to the weighted bipartite multigraphs. The extension does not change the complexity class of the problem since the problem's coefficient matrix is still totally unimodular even with a bipartite multigraph [19]. We term the algorithms the Multi-Choice (MC) Hungarian and Multi-Choice (MC) Murty's ranking algorithm.

Then we suggest polynomial-time algorithms for the L-type and C-type problems. For brevity, we denote them by the (optimal) L-type algorithm and the (approximate) C-type algorithm, respectively. The algorithms consist of two phases: the optimization phase and the rounding phase. In the first phase, we relax the integral constraint (3.5) so that a solution can be obtained in polynomial time, but it can be fractional. Thus, the second phase rounds a fractional solution to ensure the integrality of the assignment. We use an interior point method (IPM) in the first phase and the MC Hungarian method in the second phase. The L-type algorithm is optimal, and the C-type algorithm is near-optimal. We provide the performance guarantee of the C-type algorithm.

3.4.1 The multi-choice Hungarian method

We generalize the Hungarian method to allow multiple choices of performing tasks. Fig. 3.5 shows the differences in input and output between the original Hungarian method and the MC Hungarian method. For implementation, we modify the labeling operations (the initialization and the update operations) and the path augmentation from the original Hungarian method. The labeling operations include all p_{ij} edges incident to i and j. In the path augmentation step, the minimum-weighted edge among p_{ij} is selected as the path between i and j. The pseudocode is given in Alg. 1. The time complexity of this algorithm is $O(p^2(\max\{n,m\})^3)$.

Algorithm 1 The Multi-Choice (MC) Hungarian method

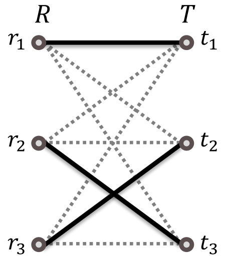
```
Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph
$G = (R, T, E)$ where $|R| = n, |T| = m$ and $p_{ij} = p, \forall\{i, j\}$.
Output: An optimal assignment $M^*$ and its cost $c^*$.

1. Generate initial labeling $l(i) = \min_{1\leq j\leq m}\{c_{ijk}\}$,
   $\forall i \in [1, n]$ and $l(j) = 0, \forall j \in [1, m]$ and matching $M$.
2. If $M$ perfect, stop. Otherwise, pick an unmatched vertex
   $r \in R$. Set $A = \{r\}$, $B = \varnothing$.
3. If $N(A) = B$, update labels by
         $l(r) = l(r) - \delta \quad r \in A$
         $l(t) = l(t) + \delta \quad t \in B$
   where $\delta = \max_{r\in R, t\in T-B}\{l(r) + l(t) + c_{ijk}\}$.
4. If $N(A) \neq B$, pick $t \in N(A) \setminus B$.
 4a. If $t$ unmatched, $u \to t$ is an augmenting path, then
     augment $M$ and go to step 2.
 4b. If $t$ is matched to $z$, extend alternating tree by
     $A = A \bigcup\{z\}$, $B = B \bigcup\{t\}$, and go to step 3.
Note: $N(r) = \{t|(r, t) \in G_e\}$, where $G_e$ is the equality graph, and $N(A) = \bigcup_{\forall r\in A} N(r)$.
```

3.4.2 The multi-choice Murty's ranking algorithm

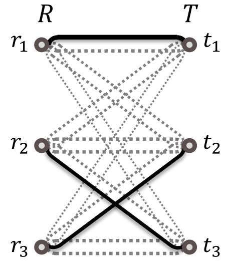
We modify the partitioning part of the original ranking algorithm. The set of all matchings is partitioned into subsets by removing each vertex and edges of s-th matching. After finding an optimal solution of each subset by Alg. 1, the vertices and the edges of the optimal solution are recovered. In the removing and recovering procedures, p_{ij} edges are removed and recovered all together. The other parts are same as the original version. The time complexity of this algorithm is $O(sp^2(\max\{n,m\})^4)$.

	t_1	t_2	t_3
r_1	4	9	6
r_2	5	11	4
r_3	6	10	5



(a) The Hungarian method solves the problems that have only single choice.

	t_1		t_2		t_3	
	p_1	p_2	p_1	p_2	p_1	p_2
r_1	5	10	6	6	6	9
r_2	5	15	7	7	14	4
r_3	6	8	9	2	5	7



(b) The MC Hungarian method allows multiple choice of performing tasks.

Figure 3.5: A comparison between the Hungarian and the MC Hungarian methods. Their input cost matrices with output assignments (shaded squares) and corresponding graphs are shown. A bold line indicates an allocation of a robot to a task. The second summation in (3.2) and (3.3) ensures a task to be performed through only one resource if $p_{ij} > 1$.

3.4.3 Exact algorithm for the P-type problem

3.4.3.1 Algorithm description

The pseudocode is given in Alg. 2. We denote the s-th assignment before/after penalization as $X^{s-/+}$ and its cost is $c^{s-/+}$. Similarly, Q^s refers the penalization of the s-th assignment. In the first iteration (i.e., $s = 1$), the algorithm computes the best assignment without penalization (c^{1-}). The penalization of the best assignment (Q^1) is computed and added to the cost of the best assignment ($c^{1+} = c^{1-} + Q^1$). Then, the algorithm computes the next-best assignment and compares its unpenalized cost (c^{s-}) with the minimum penalized cost to the previous step ($\min\{c^{1+}, ..., c^{(s-1)+}\}$). The MC Murty's ranking algorithm enables recursive computation of the next-best assignment (line 3). The algorithm repeats each iteration until either of the following conditions are met: when an unpenalized cost is greater or equal to the minimum penalized cost so that $\min\{c^{1+}, ..., c^{(s-1)+}\} \leq c^{s-}$, or the algorithm has enumerated all assignments ($N_\Pi = {}_mP_n \times \Pi_{i,j}^{n,m} p_{ij}$).

Fig. 3.6 illustrates the terminating condition of the algorithm. The algorithm computes the best ($s = 1$) assignment and its unpenalized cost. Once an assignment is determined, its penalization is computed and added to the unpenalized cost. The algorithm enumerates assignments iteratively and terminates when an unpenalized cost is larger than the current minimum cost including penalization. In the figure, the fourth assignment has a larger unpenalized cost than the cost of the second (current minimum) assignment. Thus, the algorithm terminates after it computes the unpenalized cost of the fourth assignment. Even without penalizations, all subsequent assignments have larger unpenalized costs than the minimum cost. The exact algorithm guarantees optimality but has potentially impractical running-time, as it may enumerate factorial numbers (N_Π) of iterations in the worst case.

Algorithm 2 Exact algorithm

Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph $G = (R, T, E)$ where $|R| = n, |T| = m$ and $p_{ij} = p, \forall\{i, j\}$, and penalization functions Q_l for all l.
Output: An optimal assignment X^* and its cost c^*.

```
1  Initialize $s = 1$
2  **while** $s < N_\Pi$
3      Compute $X^s$ and $c^{s-}$ //MC Murty's ranking algorithm
4      **if** $s = 1$
5          Compute $Q^s$ and $c^{s+} = c^{s-} + Q^s$
6          $s = s + 1$
7      **else**
8          **if** $(c^{s-} \geq \min\{c^{1+}, ..., c^{(s-1)+}\})$
9              $X^* = X^{s-1}$ and $c^* = \min\{c^{1+}, ..., c^{(s-1)+}\}$
10             **return** $X^*$, $c^*$
11         **else**
12             Compute $Q^s$ and $c^{s+} = c^{s-} + Q^s$
13             $s = s + 1$
14         **end if**
15     **end if**
16 **end while**
17 $X^* = X^s$ and $c^* = \min\{c^{1+}, ..., c^{s+}\}$
18 **return** $X^*$, $c^*$
```

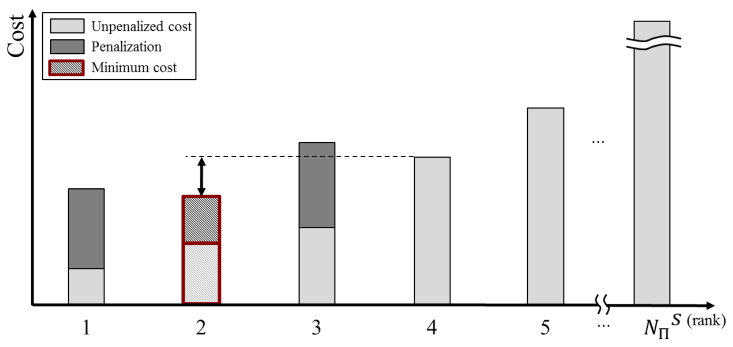


Figure 3.6: An illustration of the exact algorithm's terminating condition. When an unpenalized cost is larger than the current minimum cost (including a penalization), at $s = 4$, the algorithm terminates because all subsequent assignments cost more than the minimum cost even without penalizations.

3.4.4 Optimal algorithm for the L-type problem

The first phase uses an interior point method (IPM) for linear programming (LP). LP has the optimal solution on a vertex of a polytope. All vertices of a polytope defined by a TU matrix are integer. However, an IPM may produce a fractional solution in which a problem has multiple optimal solutions [30]. In this case, all optimal solutions form an optimal face of the polytope [109]. It is then likely that an IPM converges to an interior point of this optimal face, which is not integer. By using an IPM, we obtain a polynomial running time[7] but lose the integrality of the solution.

If the solution from the first phase is fractional, we use the MC Hungarian method to choose one of the multiple optimal solutions which is integer. The fractional matrix from the first phase is doubly stochastic: the sum of each value in a row and a column is equal to one (e.g., $\left(\begin{smallmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{smallmatrix}\right)$). A doubly stochastic matrix must be produced because the assignment satisfies the mutual exclusion constraint (3.2)–(3.3), which is same with the definition of the doubly stochastic matrix. Owing to the combinatorial structure of the fractional

[7]The simplex method does not produce a fractional solution because it visits only vertices which lie on integer points. However, the simplex method is not a polynomial-time algorithm because it could visit exponentially many vertices.

assignment matrix, each value of the assignment variables can be interpreted as a weight of the likelihood where the variable has the value of one. We use the MC Hungarian method where the input matrix (i.e., cost matrix) is the fractional assignment matrix. The MC Hungarian method for rounding outputs an integer assignment matrix (satisfying the mutual exclusion and the integral constraints) whose cost sum is the maximum. Therefore, the fractional matrix is combinatorially rounded (e.g., $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$). The pseudocode of the L-type algorithm is not given due to the space limit but same with Alg. 3 except Line 1: it uses an IPM for LP.

The time complexity of the IPM for LP that we used is $O((\max\{2n, nmp\})^3 L)$ [47][8] where L is the bit length of input variables. The Multi-Choice Hungarian method has $O(p^2(\max\{n, m\})^3)$ complexity. Thus, the overall time complexity is $O((\max\{2n, nmp\})^3 L)$. We use MOSEK optimization toolbox for MATLAB [80], particularly `msklopt` function.

3.4.5 Approximation algorithm for the C-type problem

The pseudocode is given in Alg. 3. The first phase uses an IPM for a convex optimization problem. The objective function must be twice differentiable to use the IPM. In convex programming, the solution could be fractional because not only are there multiple optimal solutions but also it is the unique optimal fractional solution. We also use the MC Hungarian method to round fractional solutions. Since the rounded solution may not be an optimal integer assignment, we provide its performance guarantee.

Theorem 3.8 The performance guarantee of the C-type algorithm is $\max\{Q_{1,\ldots,N_\Pi}\} - \min\{Q_{1,\ldots,N_\Pi}\}$.

Proof. Let $P_{1,\ldots,N_\Pi}$ be assignments of an C-type problem instance and $Q_{1,\ldots,N_\Pi}$ be the penalizations of the assignments. Let K be the upper bound of unpenalized costs, so all assignments can have their unpenalized costs up to K. Without loss of generality, all

[8]The state of the art is [4] whose complexity is $O(\frac{\max(2n,nmp)^3}{\ln \max(2n,nmp)} L)$.

$P_{1,\ldots,N_\Pi}$ have the unpenalized cost K because there can be multiple assignments that have the same cost sum. Let J be the largest integer solution among $P_{1,\ldots,N_{\Pi-1}}$ and P_{N_Π} be the optimal assignment whose cost is J^*. Then

$$J = \mathrm{K} + \max\{Q_{1,\ldots,N_{\Pi-1}}\},$$

and we define

$$J^* = \mathrm{K} + \epsilon + Q_{N_\Pi}$$

where ϵ is a nonnegative real number.

Since $J \geq J^*$, $\max\{Q_{1,\ldots,N_{\Pi-1}}\} \geq Q_{N_\Pi}$ which means $Q_{N_\Pi} = \min\{Q_{1,\ldots,N_\Pi}\}$. Then

$$\begin{aligned} J - J^* &= J - (\mathrm{K} + \epsilon + Q_{N_\Pi}) = J - \mathrm{K} - \epsilon - Q_{N_\Pi} \\ &\leq J - \mathrm{K} - Q_{N_\Pi} = \max\{Q_{1,\ldots,N_{\Pi-1}}\} - Q_{N_\Pi} \\ &= \max\{Q_{1,\ldots,N_{\Pi-1}}\} - \min\{Q_{1,\ldots,N_\Pi}\}. \end{aligned}$$

Since $\max\{Q_{1,\ldots,N_\Pi}\} \geq \max\{Q_{1,\ldots,N_{\Pi-1}}\}$,

$$J - J^* \leq \max\{Q_{1,\ldots,N_\Pi}\} - \min\{Q_{1,\ldots,N_\Pi}\}.$$

$\square$

The significance of the performance guarantee can differ depending on the viewpoint. Clearly, the importance of the performance guarantee depends on the precise form (and the value) of the penalization function. Thus, the guarantee is less important when its particular value is very large. However, the guarantee is significant in the sense of its independence from assignments. Regardless of which assignment is computed, the guarantee solely depends on the penalization function. We have seen that the hardness of the problem has a spectrum depending on the form of the penalization function (in Section 3.3.5). Likewise, the importance of the guarantee also has a spectrum depending on the form of the penalization function. If a penalization function has bounded changes with respect to its input, the difference between an approximation and an optimal value would not be very large, so the (practical) importance of the guarantee is more significant. We may construct a penalization function that satisfies a certain condition (e.g., limiting the function's change), and increase the significance of the performance guarantee.

The time complexity of an IPM for a convex optimization problem is $O((\max\{2n, nmp\})^{3.5}L)$ [85]. Thus, the overall time complexity is $O((\max\{2n, nmp\})^{3.5}L)$. We use MOSEK `mskscopt` function for the optimization phase. Table 3.4 summarizes the problems and algorithms.

Table 3.4: A summary of the problems and algorithms.

Problem:		P-type	C-type	L-type
Objective function		Polynomial-time computable	Convex	Linear
Complexity class		NP-hard	NP-hard	P
Algorithm	Step I	Iterative method (Ranking alg. + MC Hungarian)	Linear programming (IPM)	Convex optimization (IPM)
	Step II		Rounding (MC Hungarian method)	
Overall complexity		$O({}_mP_n \times \Pi_{i,j}^{n,m} p_{ij})$	$O((\max\{2n, nmp\})^{3.5}L)$	$O((\max\{2n, nmp\})^{3}L)$
Performance guarantee		Optimal	$\max\{Q_{1,\ldots,N_\Pi}\} - \min\{Q_{1,\ldots,N_\Pi}\}$	Optimal

3.5 Extension: Modeling Synergies

The modeling method presented in this chapter also can be applied to modeling negative penalization, namely synergies. Some synergistic effects make interrelations among costs and can be modeled by a concise representation like $Q(\cdot)$ in (3.1). Fig. 3.7 shows an example. Robots are located outside of a cluttered disaster site (e.g., a collapsed building or an explosion site). The robots have individual tasks (e.g., monitoring survivors until assistance arrives) inside of the site. There are multiple paths to reach the tasks, but they

Algorithm 3 The C-type algorithm

Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph $G = (R, T, E)$ where $|R| = n, |T| = m$, $p_{ij} = p, \forall\{i, j\}$, and convex penalization functions Q_l for all l.
Output: An optimal assignment X^* and its cost c^*.

```
1 Compute $X^*_{\mathbb{R}+}$ and $c^*_{\mathbb{R}+}$ //IPM for CP
2 Compute $\hat{X}^*_{\mathbb{Z}+}$ and $\hat{c}^*_{\mathbb{Z}+}$ //MC Hungarian method
3 $X^* = \hat{X}^*_{\mathbb{Z}+}$, $c^* = \hat{c}^*_{\mathbb{Z}+}$
4 return $X^*$, $c^*$
```

are covered with debris. Robots should push their ways through the debris. The goal is to minimize the total traveling time to reach their destinations. Even though the robots perform their own tasks, there can be collaborations among robots on the same resource such as pushing debris together on the same path. Robots are neither tightly coupled to make coalitions nor required to have pre-coordination. Collaborations make synergistic effects, but more robots on the same resource produce more resource contention so the number of robots on shared resources is needed be determined optimally. Note that the exact algorithm is not applicable when negative costs may be incurred by synergistic effects. In this case, the negative costs could make any subsequent penalized cost c^{s+} less than the current optimal assignment c^*, so the algorithm is not able to decide whether to terminate before it enumerates all assignments. However, the other two algorithms are still applicable when synergistic effects are modeled as a linear or a convex function.

Applications are not limited to physical interactions. When robots explore to search for their individual targets, one robot can connect to a network and tell other robots in the network what it detects if the detected target is not the robot's target. The collaboration reduces searching time and its effectiveness is amplified as more robots participate by connecting to the same network. However, this synergistic effect would be deteriorated as more robots connect to the same network because the communication bandwidth is

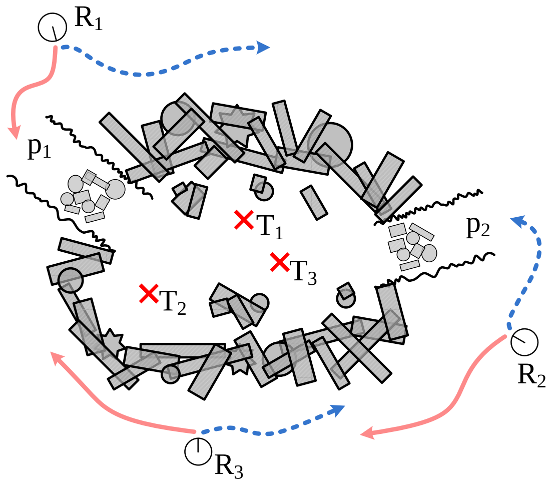


Figure 3.7: An example in which both synergy and resource contention occur while robots perform individual tasks. Tasks are inside of the cluttered disaster site, and each robot can choose one path between p_1 and p_2 to reach the tasks. The robots who choose the same path become to push debris together.

limited.

3.6 Experiments

We demonstrate that the exact algorithm works well and returns a result in reasonable time for practically sized cost matrices. The L-type and C-type algorithms produce solutions quickly for even larger matrices. We implemented all the algorithms in MATLAB. The solution quality is measured by a ratio of an approximated solution to an optimal solution $\eta = \frac{c'^*}{c^*} \geq 1$. We assume that $n = m$ and $p_{ij} = p, \forall \{i.j\}$ for all the experiments. If the p_{ij}s are not identical, then we add dummy edges with infinite cost. As we detail next, both randomly generated problem instances and instances based on real-world scenarios were used to validate the algorithms. Also, they are demonstrated with physical robots in small-scale experiments in our laboratory. First, we provide some detail on the particular penalization models used.

3.6.1 Penalization functions

A penalization function models the interference incurred in a particular environment, and should consider the specific aspects of the robots and environment. Simple examples based on a factorization that adds costs as a function of the number of robots utilizing a resource, include models in the form of linear and an convex quadratic functions. Following the form in (3.6), let those penalization models for use of the l-th resource be

$$Q_l(n_l) = \begin{cases} \beta_{\mathrm{L}} n_l + \beta'_{\mathrm{L}} & n_l \geq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3.13}$$

and

$$Q_l(n_l) = \begin{cases} \beta_{\mathrm{C}} n_l^2 + \beta'_{\mathrm{C}} n_l + \beta''_{\mathrm{C}} & n_l \geq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3.14}$$

where β_{L}, β'_{L}, β_{C}, β'_{C}, and β''_{C} are constants.

For the multi-vehicle transportation scenario, we used the classic flow model developed by domain experts to quantify traffic congestion [86]. Many models have been proposed in the literature that compute a traffic speed v (m/s) according to traffic density ρ (vehicle/m). We let $\rho = n_l$ because we only consider an instantaneous assignment problem. We use an exponential model for our application that travel time (sec) is used as cost

$$Q_l(n_l) = \begin{cases} \frac{d_l}{v_{\mathrm{f}}\left[1-\exp\left\{-\frac{\lambda}{v_{\mathrm{f}}}\left(\frac{1}{n_l}-\frac{1}{\rho_j}\right)\right\}\right]} & \rho_j \geq n_l \\ +\infty & \text{otherwise,} \end{cases} \tag{3.15}$$

where v_{f} is the free flow speed (when $n_l = 0$), ρ_j is the jam density, and λ is the slope of the headway-speed curve[9] at $v = 0$, and d_l is the length of a resource that could be shared with other robots such as a passage.

We also suggest an idea of designing wireless network congestion models. Throughput of per client (Mbps) with respect to the number of client (n) is shown in [37, Figure 4 and Table 4]. From the data, a quadratic convex function is fitted where the function represents the relationship between the number of clients (robots) and the unit transmission time (reciprocal of throughput). This is a rough modeling based on the data sheet, and domain experts may build more accurate models. The network congestion example suggested in Section 3.2.4 can use this model to minimize the total completion time including data transmission time.

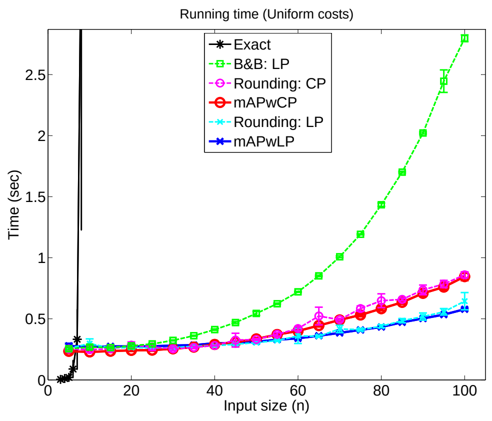
3.6.2 Random problem instances

A uniform cost distribution ($\mathcal{U}(0, 60)$) is used to test the algorithms. The penalization function (3.14) is used for the exact and C-type algorithms, and (3.13) is used for the L-type algorithm ($\beta_\text{L} = \beta_\text{C} = 1$ and $\beta'_\text{L} = \beta'_\text{C} = \beta''_\text{C} = 0$). With fixed $p = 5$, the size of the cost matrix (n) increases from 5 to 100 at intervals of 5 (10 iterations for each n). Fig. 3.8 and Table 3.5 show running times and solution qualities of our algorithms. We also compare them with conventional methods such as branch and bound (BB) and randomized rounding.[10] We do not report results for the BB method on the C-type because the running-time is impractical and prohibitive even when the instance size is small (e.g., $n = 10$). We display results in multiples of 10, owing to the space limitations.

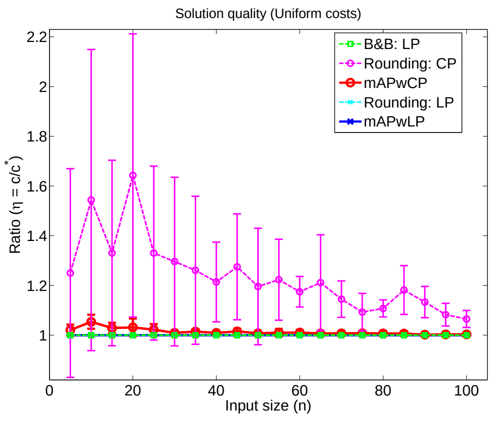
The exact algorithm finds an optimal assignment in a reasonable time on small instances ($n \leq 8$); even a small problem has a huge search space (e.g., $N_\Pi = 375,000$ when

[9]The ratio of (infinitesimal) velocity change over (infinitesimal) headway change.

[10]If a penalization function is linear, network flow algorithms can have a better bound (e.g., $O(pn^3)$) than the L-type algorithm. For convex penalization functions, network flow algorithms have no way to deal with nonlinear objective functions [11]. One of the possible ways that leads to global optimality for the problems with nonlinear objective functions is to use a nonlinear programming formulation. However, the formulation may not produce integer solutions so BB and randomized rounding are used.



(a) Running times



(b) Solution qualities

Figure 3.8: Running time and solution quality of random instance. (a) The L-type and C-type algorithms are slightly faster than the rounding method whose worst case running time is exponential. (b) The C-type has better solution quality than the rounding method.

$n = 5$ and $p = 5$). The L-type and C-type algorithms quickly find solutions even if n is large. Our methods are faster than the BB methods and similar to the randomized rounding methods. However, the methods we propose are the only to have polynomial running time. The solution qualities of the C-type is better than the BB method (also the proposed algorithms have a performance guarantee). The BB methods find an optimal solution but have exponential worst-case time complexity. These properties are shown in the results: the running time is longer but the solution quality is optimal ($\eta = 1$). The randomized rounding methods are faster because they have a single random-rounding step, but the randomness makes their solution quality bad.

3.6.3 Multi-vehicle transportation problems

A multi-vehicle transportation problem is used as a representative real-world application for our algorithms. We assume that n homogeneous robots and n tasks are distributed across p bridges in an urban area as shown in Fig. 3.9. The robots and the tasks are uniformly distributed within the boundaries. Distances from the robots to the tasks though the bridges are collected by using the Google Directions API [48]. The raw data are in me-

Table 3.5: Running time and solution quality of random instances.

(a) The exact algorithm.

	n	3	4	5	6	7	8	9
Running time (sec)	Mean	0.0041	0.0115	0.0208	0.0894	0.3324	3.8327	95.1580
	Std. dev.	0.0043	0.0047	0.0144	0.0721	0.2467	2.6072	93.7848

(b) The L-type and C-type algorithms and existing methods.

		n	10	20	30	40	50	60	70	80	90	100
L-type	Running time (sec)	Mean	0.2689	0.2743	0.2812	0.3003	0.3127	0.3406	0.3848	0.4333	0.5029	0.5786
		Std. dev.	0.0040	0.0091	0.0037	0.0064	0.0062	0.0046	0.0059	0.0081	0.0091	0.0067
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
C-type	Running time (sec)	Mean	0.2296	0.2421	0.2546	0.2898	0.3354	0.4005	0.4908	0.5835	0.7094	0.8462
		Std. dev.	0.0051	0.0068	0.0055	0.0044	0.0071	0.0116	0.0101	0.0150	0.0187	0.0311
	Quality (η)	Mean η	1.0537	1.0314	1.0093	1.0092	1.0076	1.0104	1.0074	1.0067	1.0020	1.0030
		Std. dev.	0.0282	0.0353	0.0086	0.0078	0.0042	0.0105	0.0089	0.0100	0.0016	0.0020
B&B: LP	Running time (sec)	Mean	0.2688	0.2787	0.3235	0.4121	0.5442	0.7207	1.0080	1.4342	2.0219	2.7971
		Std. dev.	0.0160	0.0084	0.0040	0.0054	0.0162	0.0037	0.0070	0.0137	0.0080	0.0277
	Quality (η)	Mean η	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Rounding: LP	Running time (sec)	Mean	0.2972	0.2662	0.2644	0.2797	0.3051	0.3598	0.4173	0.4392	0.5186	0.6446
		Std. dev.	0.0386	0.0146	0.0072	0.0053	0.0067	0.0619	0.0305	0.0121	0.0295	0.0706
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Rounding: CP	Running time (sec)	Mean	0.2477	0.2841	0.2538	0.2816	0.3310	0.4203	0.4930	0.6485	0.7368	0.8602
		Std. dev.	0.0208	0.0266	0.0150	0.0041	0.0075	0.0164	0.0104	0.0564	0.0398	0.0249
	Quality (η)	Mean η	1.5435	1.6424	1.2960	1.2140	1.1957	1.1746	1.1448	1.1075	1.1332	1.0651
		Std. dev.	0.6057	0.5695	0.3391	0.1603	0.2342	0.0614	0.0731	0.0342	0.0630	0.0342

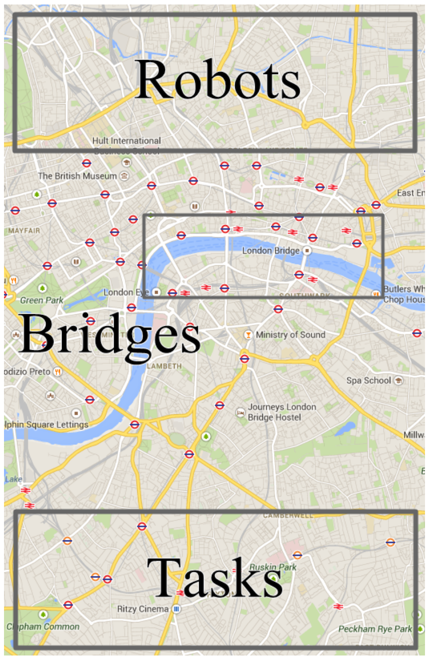


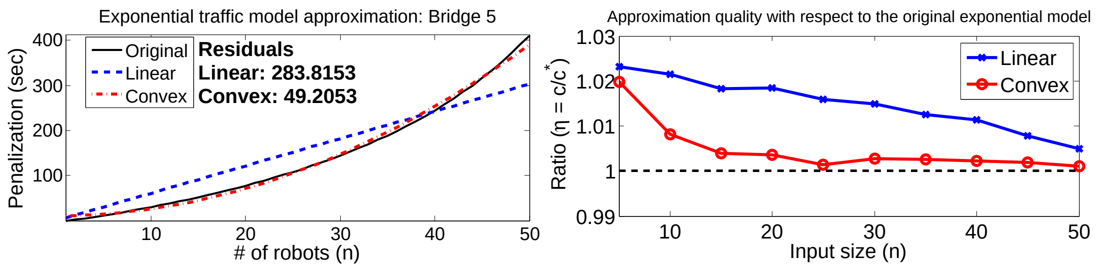
Figure 3.9: Robots and tasks are located across five bridges. n robots and tasks are uniformly distributed in the upper and lower boxes, respectively.

ters (m) but converted to time (sec) according to v_{f}. Thus, the cost is travel time without congestion and penalized by the increased time owing to congestion. With fixed $p = 5$, n increases from 5 to 50 (3 to 9 for the exact algorithm). Other parameters are set as follows:

$$\begin{aligned}
v_{\mathrm{f}} &= 16.67 \text{ m/s}, \\
d_l &= \{500,\ 300,\ 250,\ 400,\ 200\} \text{ m}, \\
\rho_{jl} &= \{120,\ 80,\ 70,\ 90,\ 80\} \text{ robot/choice} \\
\lambda_l &= \{0.1389,\ 0.1667,\ 0.1528,\ 0.1944,\ 0.1389\} \text{ s}^{-1}
\end{aligned}$$

where $l = 1, ..., 5$. The parameters reflect the characteristics of the real-world multi-vehicle transportation problem.

We use (3.15) for the exact algorithm. However, our implementations for Alg. 3 do not allow a complex exponential objective function like (3.15). Thus, we approximate (3.15)



(a) An example of approximations (b) Solution quality

Figure 3.10: Approximations of a complex nonlinear function to simple functions for practical implementations. (a) We approximate a complex exponential function with a linear and a convex quadratic function. (b) Solution qualities when the approximated functions are used for all five bridges.

with a linear and a convex quadratic function such as (3.13) and (3.14). An example of the approximations is shown in Fig. 3.10a. The quality of the approximation is measured by the sum of squared residuals. Table 3.6 shows the approximation results of all penalization functions of the five bridges.

Table 3.6: Penalization function approximation results of the five bridges.

	Residuals				
Fitting type	Bridge 1	Bridge 2	Bridge 3	Bridge 4	Bridge 5
Linear	301.2	408.8	531.1	389.6	283.8
Convex	22.92	70.61	124.1	52.28	49.21

The Fig. 3.10b shows solution qualities when the approximated functions are used. For each instance, we compute an optimal assignment with the exact algorithm when the original model is used. Then we compare it to the assignments when the approximated functions are used. As a result, the solution qualities are good (less than 1.024) so those approximations are acceptable.

We compare our method with the optimal assignment problem formulation that does not include additional costs incurred by resource contention (but the additional costs occur when robots perform tasks). We use Alg. 3 with the convex quadratic penalization function that we approximated from (3.15). For the comparison, we use the Hungarian method to compute an optimal assignment without considering penalization. Once an assignment is obtained, we use the same convex quadratic penalization function to the additional costs associated with the assignment. Table 3.7 show the results (10 iterations for each n). The results show that incorporating resource contention into this transportation problem is crucial to achieve global optimality.

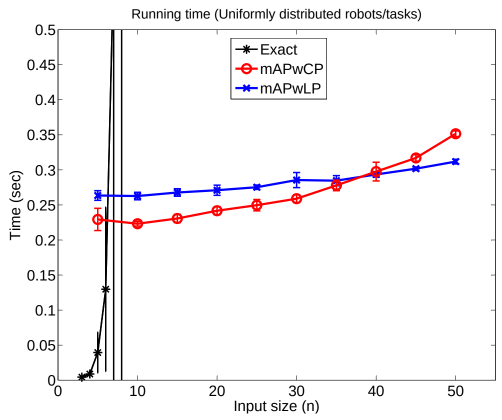
Table 3.7: Differences of the total cost sums between the C-type and the Hungarian method in the multi-vehicle transportation problem.

n	5	10	15	20	25	30	35	40	45	50
Cost difference (sec)	5.5311	15.7429	14.0122	20.4954	21.3051	40.0226	45.8332	49.9823	61.9301	83.9979

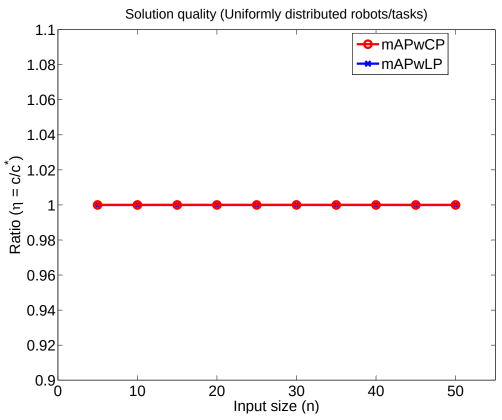
Next, we compare our algorithms with the existing methods that use our models. Fig. 3.11 and Table 3.8 show the results (10 iterations for each n). The results are similar to the random instance case. This experiment shows that our algorithms can model realistic scenarios of robotic applications.

3.6.4 Physical robot experiment

We demonstrate that our method achieves global optimality even interference is not negligible. Fig. 3.12 shows the experimental setting. Two iRobot Creates (R_1 and R_2) have tasks of visiting the other robot's position on the opposite side of environment (T_1 and T_2). There are two passages to reach their destinations (shown as p_1 and p_2 in the figure). We use travel time as the cost and (3.15) as the penalization function. We compute the assignment with Alg. 2. We assume that R_1 and R_2 are functionally identical. Space constraints and the data from the previous experiments forced us to omit reporting quantitative results.



(a) Running times



(b) Solution qualities

Figure 3.11: Running time and solution quality of the multi-vehicle transportation problem. (a) The L-type and C-type algorithms quickly produce solutions. (b) The qualities are close to one for both algorithms.

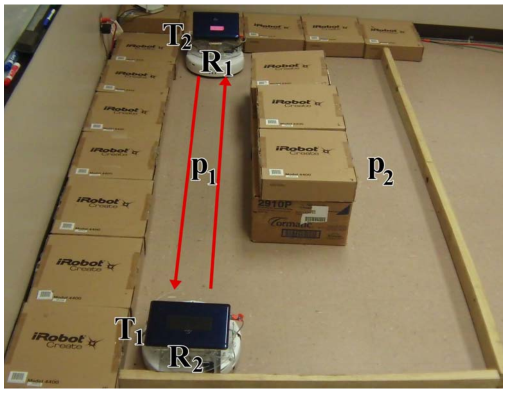
Table 3.8: Running time and solution quality of the multi-vehicle transportation problem.

(a) The exact algorithm.

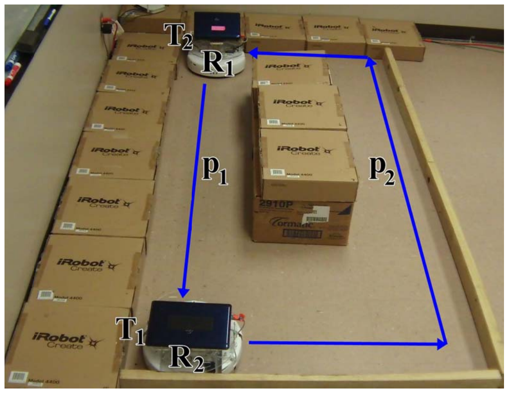
	n	3	4	5	6	7	8	9
Running time (sec)	Mean	0.0044	0.0088	0.0395	0.1298	0.5913	4.2897	126.0774
	Std. dev.	0.0015	0.0045	0.0290	0.1171	0.8772	6.8811	202.1757

(b) The L-type and C-type algorithms.

		n	5	10	15	20	25	30	35	40	45	50
L-type	Running time (sec)	Mean	0.2634	0.2627	0.2678	0.2710	0.2753	0.2855	0.2846	0.2935	0.3017	0.3118
		Std. dev.	0.0069	0.0054	0.0054	0.0073	0.0032	0.0108	0.0072	0.0037	0.0024	0.0033
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
C-type	Running time (sec)	Mean	0.2293	0.2232	0.2307	0.2417	0.2496	0.2589	0.2779	0.2976	0.3170	0.3515
		Std. dev.	0.0159	0.0034	0.0059	0.0058	0.0081	0.0056	0.0076	0.0133	0.0045	0.0040
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000



(a) Both R_1 and R_2 use p_1.



(b) R_1 uses p_1, and R_2 uses p_2.

Figure 3.12: Two cases of resource use by two mobile robots. (a) Robots use the same resource so that interference is occurred. (b) Robots use different resources to avoid the interference.

When the robots move through the shortest path to the destination to attain the minimum travel time, they choose the same passage p_1 (Fig. 3.12a). However, this choice incurs interference between the robots. When the assignment is penalized, the best assignment is changed to the other assignment: R_1 uses p_1 and R_2 uses p_2 (Fig. 3.12b). When the robots use the same resource p_1, it takes 102 seconds to complete the tasks whereas the interference-free assignment takes 87 seconds.

3.7 Summary

In this chapter, we define the mAPwP problems and show that the P-type and C-type problems are NP-hard, and the L-type problem is in P. We develop the Multi-Choice Hungarian method, which is a generalization of the original method, to allow multiple choices of performing tasks. We present an exact algorithm that generalizes Murty's ranking algorithm to solve the multi-choice problem, which employs the MC Hungarian method as a subroutine. In addition, we propose two polynomial-time algorithms for the L-type and C-type problems. The L-type algorithm produces an optimal assignment, and the C-type algorithm computes a solution with bounded quality. In the experiments, we model interference among robots by introducing several penalization functions; the results show that the exact algorithm finds an optimal solution, and the L-type and C-type algorithms produce an optimal and a high-quality solution quickly. We also conduct physical robot experiments to show how resource contention aggravates optimality in practice and that the proposed algorithm achieves global optimality when an interference model is included.

4. MITIGATING WEAKNESSES IN CENTRALIZED MULTI-ROBOT TASK ALLOCATION WITH UNCERTAIN AND INTERRELATED COSTS

In MRTA, it is general that each robot estimates the costs of performing each task, then these estimates are shared by robots over a communication network. Most often an optimal assignment is computed by a central computation unit (e.g., most approaches using the Hungarian method [65], an auctioneer [34], or a linear programming framework). But while robots are executing their assigned task, the assumptions under which costs were calculated may turn out to be invalid: the environment may change, robots may fail, or a variety of other unexpected situations may emerge. One solution which ensures a fluid response to these contingencies, is to periodically re-calculate costs and re-compute the optimal task assignments. This solution incurs computational and communication expense proportional to the desired recency.

We are interested in the MRTA problem where an instantaneous scalar estimate of a cost may be inappropriate or invalid. This arises naturally when there is uncertainty in some state used in computing the costs, or when the costs evolve as tasks are performed and the estimates are out of date. The first and perhaps most straightforward representation for an uncertain cost is to generalize a single value to a range of possible values, moving geometrically from a point to a line. For example, the autonomous robot in Fig. 4.1 is able to estimate its shortest and longest driving times (which may be used as cost measures) to a destination by considering information about its route. The lower bound of the time would be merely the time spent on driving (distance over the maximum speed), and adding the maximum waiting time for traffic signals yields the upper bound[1]. Then, conceptually, all

© 2015 IEEE. Reprinted, with permission, from Changjoo Nam and Dylan Shell, "When to do your own thing: Analysis of cost uncertainties in multi-robot task allocation at run-time," in *Proceedings of IEEE International Conference on Robotics and Automation*, May 2015.

[1]For simplicity, we assume an absence of congestion and acceleration/deceleration here.

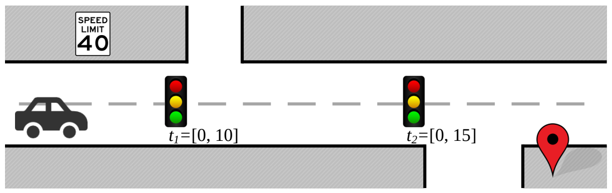
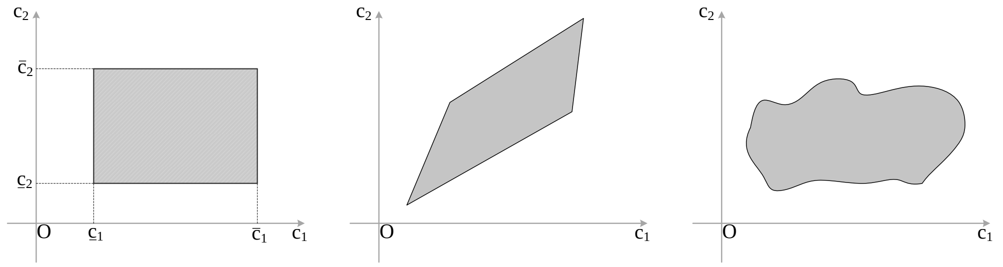


Figure 4.1: A simple example where task costs which are not precisely known to the robot beforehand. The driving time c to the destination will vary depending on the traffic signals. A lower bound $\underline{c}$ is $\frac{d}{v_{\max}}$ when $t_1 = t_2 = 0$, and an upper bound $\bar{c}$ is $\frac{d}{v_{\max}} + 25$ (assuming the robot drives with the maximum speed) where d is the distance to the destination and $v_{\max}$ is the maximum speed.



(a) A cost region defined by an upper and a lower bound. Costs are uncertain but independent.

(b) A linear convex boundary showing both uncertainty and a linear interrelationship.

(c) A nonlinear nonconvex boundary, for complex uncertainty and interrelationships between costs.

Figure 4.2: Figurative illustrations of the region-based cost representation. Costs that are uncertain and interrelated are represented with boundaries by treating the set of possible costs as regions.

permissible values for costs fall in a (high-dimensional) region as illustrated in Fig. 4.2a. Regions more complex than an axis-aligned box can describe nontrivial interrelationships between the permissible costs (as in Fig. 4.2b or Fig. 4.2c).

Sensitivity analysis (SA) has been studied for several decades in Operations Research to assess the robustness of optima for an optimization problem to perturbations in the input specification [38], [104], [70]. Analysis of an optimal assignment must compute a region where costs within that region preserve the current optimal assignment. However, SA has found limited applicability to multi-robot task-allocation problems: the analysis assumes that the decision maker (its counterpart in multi-robot systems is the central computation unit) is able to access all information off-line and has control all over the constituents without communication constraints. The physically distributed nature of multi-robot systems and their limited communication and computational resources pose challenges to the direct application of classical SA.

We use ideas from SA to develop methods that explore questions pertinent to resource limitations in multi-robot systems. For a given problem instance, these methods reduce global communication and centralized computation, or quantify the optimality trade-offs if communication is avoided. This chapter makes the following contributions:

- We propose a region-based cost representation that captures the uncertainty in the states of robots, tasks, or the environment. This representation does not make the simplifying assumption where costs are independent; it models tightly interrelated costs, which enables a region of costs to present richer information.

- We develop an algorithm that analyzes the cost structure for a given assignment. It seeks cliques in the team, factorizing the group into sub-teams that are able to work independently, communicating only among themselves, forgoing global communication but without sacrificing global optimality.

- We consider the problem of deciding whether it is beneficial to persist with the current assignment even if cost changes mean that it is no longer optimal. We develop a method for computing the worst-case cost sum if the robots retain their current assignment, allowing one to decide whether to persist with the current assignment because the computational/communication expense needed for re-assignment is prohibitive.

- We examine how, once costs change, the robots can determine whether the current task assignments are sub-optimal with minimal communication. Each robot may compute a safe (one-dimensional) interval within which any cost variation does not affect optimality. But even if a cost violates these bounds, other costs may have changed too, and optimality may still be retained when the cost changes are considered together. We introduce a method that incrementally increases the dimensionality of the bounding region, growing the number of costs considered by communicating with adjacent robots. Global communication may be required in the worst case but oftentimes local computation can reach the conclusion that the assignment is still optimal.

4.1 Related Work

Some authors have proposed reoptimization schemes for multi-robot systems, allowing updated assignments to be computed efficiently. Mills-Tettey et al. [79] describe a dynamic (or incremental) Hungarian method that repairs initial optimal assignment to obtain a new optimal assignment when costs have changed. Also, Shen and Salemi [98] present a decentralized dynamic task allocation algorithm that uses a heuristic searching method. These algorithms still use computational resources for those cost modifications which end up with the same assignment.

Parker et al. [89] propose a decentralized algorithm to minimize the maximum cost where cost changes over time. They represent a cost as a monotonically increasing function as time passes (e.g., fire spreading). Each agent assigned to a task decreases the cost with a fixed rate. They propose a modified MAX-SUM algorithm that optimizes a global utility function in a greedy way where the modification of the original max-sum algorithm is made to incorporate uncertainty of the global utility. They precisely model varying costs and include the model to the max-sum framework with the notion of uncertainty. However, their method has limited applicabilities when the change of costs cannot be foreseen. Moreover, they assume that robots work with a constant rate, but the assumption could be incorrect when the performance of robots depends on exogenous factors.

Liu and Shell [71] propose the interval Hungarian method (IHM) to manage uncertainties in costs. Given an optimal assignment, the algorithm computes the maximum interval of each cost in which costs within the interval do not change the current optimal assignment. Thus, robots are able to decide how a cost change affects the optimality of the current solution. However, the algorithm treats the problem of multiple cost modifications, which do occur naturally in multi-robot systems (e.g., a single robot failure affects n costs), in an *ad hoc* fashion. The same authors also propose a sparsification and partitioning method to distribute the assignment problem to reduce global communication and re-assignment [72]. The method coarsens the utility matrix by using locality and sparsity of tasks. Once the matrix is partitioned into several clusters, each cluster is able to compute an assignment independently. Inspired by that work, we propose a factorization method for problems where mere single time-step sparsity is not enough.

4.2 Interrelated Costs in a Finite Region

In this section, we formalize the cost representation in terms of a bounded region. Then we propose three methods that alleviate dependencies on centralized structures such as global communication and centralized computation. First, given a model of how costs may evolve, we develop an algorithm that partitions a team of robots into several independent cliques, which can maintain global optimality by communicating only amongst themselves. Second, we propose a method for computing the worst-case cost sub-optimality if robots persist with the initial assignment and perform no further communication and computation. Lastly, we develop an algorithm that assesses whether cost changes affect the optimality of the current assignment through a succession of local checks.

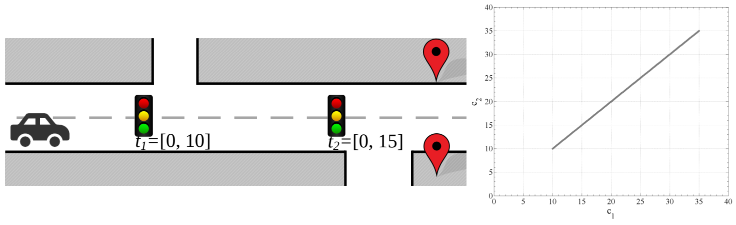
4.2.1 The representation

We assume, most generally, that the costs belong to a finite region $\mathbb{C}$ where $\mathbb{C} \subseteq \mathbb{R}^{(n^2)}$. So any cost matrix $\mathbf{C} \in \mathbb{C}$. Domain knowledge permits specification of the boundary of $\mathbb{C}$. If only upper and lower bounds of costs are known, we can define the largest cost boundary as in Fig. 4.2a, that is, as a hyper-cuboid[2]. In other words, $\mathbf{C} \in \mathbb{C}$ has for each c_{ij} a range $\underline{c}_{ij} \leq c_{ij} \leq \bar{c}_{ij}$. It serves as a concise characterization of uncertain costs but, crucially, fails to capture any interrelationships between costs.

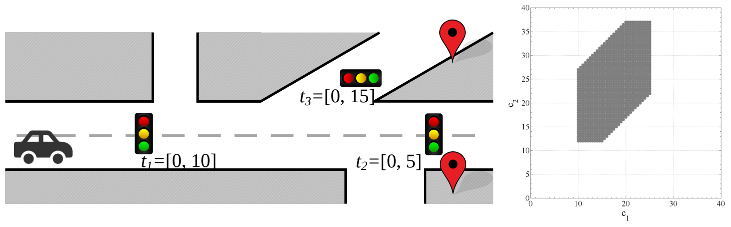
Costs that have interrelationships can be modeled by having a more complex boundary for $\mathbb{C}$ as shown in Fig. 4.3a where the boundary is modeled as a linear function. A slightly richer example appears in Fig. 4.3b where part of a route is shared between two destinations but after a fork in the road there are different waiting times and distances. The resultant cost boundary is a convex polygon.

But practically, costs could have rather more complex boundaries. Fig. 4.3c shows an example of Zermelo's navigation problem [108]. Suppose an underwater vehicle capable of navigating a certain maximum speed and heading angle moves through a water flowing with a current. The problem is to find the time-optimal path from a position to a destination. If the vehicle solves the problem by steadily aiming at the appropriate fixed angle to the current, the vehicle is able to navigate along a straight path to the destination, which is the optimal. Costs change with the change of the direction of the current. If the current may have any direction, the corresponding boundary of the costs is nonlinear convex as shown in the right graph in Fig. 4.3c.

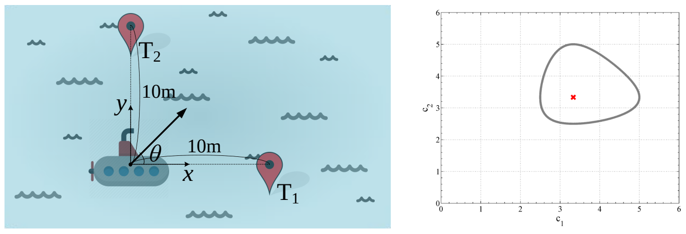
[2]The dimensionality of the cost space of an MRTA problem is often extremely high. The 2-D representation is a presentation aid.



(a) (Left) Two destinations have routes that share a common segment. If cost is proportional to driving time, the traffic signals (with unknown state) induces a delay that affects both. (Right) The line represents the possible costs.

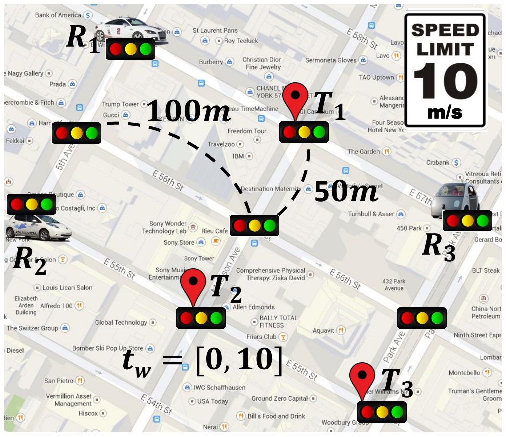


(b) (Left) A road route forks and each fork has different waiting times and lengths. One traffic signal affects both driving times to the destinations but other signals influence the times independently. (Right) The corresponding convex linear cost boundary.



(c) (Left) An underwater vehicle plans to navigate to each of two destinations. The vehicle is influenced by water current and adjusts its heading to cancel out the current. The costs depending on the angle θ of the current. (Right) The corresponding convex nonlinear cost boundary when $0 \leq \theta < 2\pi$ and the vehicle moves at the maximum speed.

Figure 4.3: Further robot navigation scenarios and corresponding cost boundaries.



(a) A multi-robot navigation example.

	T_1	T_2	T_3
R_1	$[10, 30]$	$[20, 60]$	$[30, 80]$
R_2	$[20, 60]$	$[10, 30]$	$[20, 50]$
R_3	$[10, 30]$	$[20, 60]$	$[10, 40]$

(b) The cost matrix corresponding to (a).

Figure 4.4: An example of an MRTA problem with changeable costs. We have three robots ($R_{1,2,3}$) and three destinations ($T_{1,2,3}$). The goal is to have one robot at each destination while minimizing the total sum of traveling time. Since the costs could vary within the ranges in (b), there are multiple assignments possible. The proposed methods can be used concurrently to analyze the assignments and to have less centralized operations.

4.2.2 The algorithms mitigating the weaknesses of centralized systems

With a centralized system, computing the results of the SA for the multi-robot task assignment is straightforward. The central unit computes an optimal assignment with the latest costs and (2.7). Robots then report cost changes to the central unit and which then checks if they violate $\theta(\mathbf{X}^*)$. If the change does not alter the current optimal assignment, the team keeps working as before (no other computation is needed, no other robots need be notified of the cost change).

The centralized approach is simple to implement but often undergoes problems arising from the distributed nature of multi-robot systems. Especially, maintaining global connectivity in multi-robot systems is expensive, and the quality of communication changes drastically [88]. We aim to develop methods for distributing the assignment problem to alleviate dependence on the centralized structure: (i) factorizing a team of robots into cliques if such cliques exist (Fig. 4.5a), (ii) computing the cost difference between the worst-case cost sum (if the robots persist their initial assignment) and the best-case cost sum (if they reassign tasks) (Fig. 4.5b), and (iii) communicating locally to decide whether a re-assignment is necessary with cost changes (Fig. 4.5c).

In (i), we find all N possible assignments with a cost matrix $\mathbf{C}$, which is changeable, to factorize a team of robots without locality and sparsity of tasks. The challenge is how to find N assignments. A brute-force method is computing all assignments for all costs in the hyper-cuboid, but it is impossible because there is an infinite number of $c_{ij} \in [\underline{c}_{ij}, \bar{c}_{ij}]$.

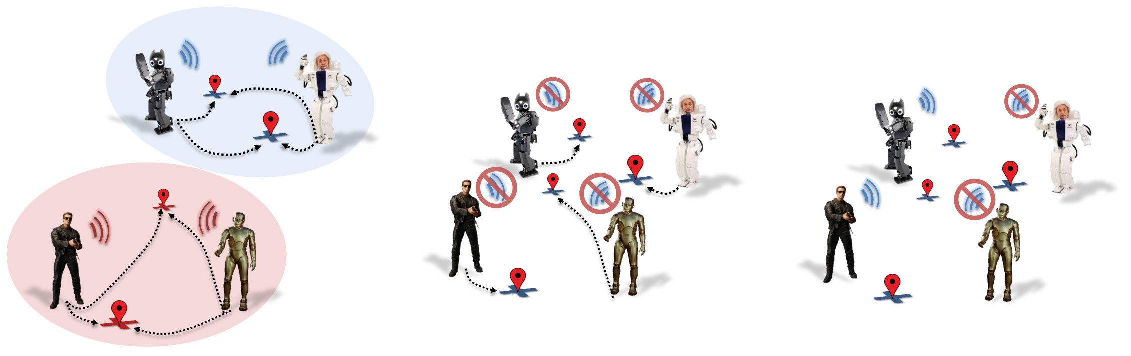
Once optimal assignments $\mathbf{X}_q^*$ and their $\theta(\mathbf{X}_q^*)$ for $q = 0, \cdots, N-1$ are computed by resolving the challenge in (i), (ii) can be solved by finding the minimum cost matrix $\mathbf{C}_{\min_q}$ in each $\theta(\mathbf{X}_q^*)$ and compute

$$\max(\bar{\mathbf{C}} \odot \mathbf{X}_0^* - \mathbf{C}_{\min_q} \odot \mathbf{X}_q^*) \tag{4.1}$$

for $q = 0, \cdots, N-1$ where the operator $\odot$ means the sum of all elements in the Hadamard product of two matrices[3]. $\mathbf{X}_0^*$ indicates the initial optimal assignment. In other words, (4.1) is the cost difference between the minimum among cost sums, where robots change their assignments for the recent cost updates, and the maximum cost sum if robots maintain their initial assignment.

If robots have one-dimensional intervals of their costs in which any cost change within its interval does not alter the current assignment regardless of other cost changes, the robots can work independently until any of their own intervals is violated. In (iii), such intervals should be computed and distributed to robots. If a robot finds one of its intervals is violated, the robot checks itself whether $\theta(\mathbf{X}_q^*)$ is violated by looking at its all other costs. If other cost changes countervail the violation, the current assignment is preserved. Otherwise, the robot communicates with an adjacent robot to consider more cost changes.

[3]Formally, $\mathbf{A} \odot \mathbf{B} = \mathbf{e}^T(\mathbf{A} \circ \mathbf{B})\mathbf{e}$.



(a) Factorizing a team into smaller cliques.

(b) Persisting an initial assignment.

(c) Local communication to check optimality violations of costs.

Figure 4.5: The methods to mitigate centralization in MRTA. (a) Cliques could be found by analyzing $\theta(\mathbf{X}_q^*)$ for $q = 0, \cdots, N-1$, where N is the number of all possible assignments with given $\mathbf{C}$. (b) The maximum cost loss is computed for which robots do not have communication and persist an initial assignment even with cost changes. (c) Robots have local communication to check whether their changed costs violate the current $\theta(\mathbf{X}^*)$.

Global communication may be required in the worst case, but local communication is often enough.

Next, we show a simple scenario and how the proposed methods can be used concurrently. After that, we briefly describe our implementations of computing (2.7) and propose three methods for the problems stated above.

4.2.2.1 An example scenario

We consider a multi-robot navigation problem. Suppose we have three autonomous robots ($R_{1,2,3}$) and three destinations ($T_{1,2,3}$) as shown in Fig. 4.4. Time is the measure of cost. The goal is to have one robot at each destination while minimizing the total sum of traveling times. We assume that the robots drive through the shortest path, and each intersection has a traffic signal. The waiting time at each signal is $t_w \in [0, 10]$. Again, we assume that robots drive at the maximum speed (10 m/s) when they move, and we do not model delays from congestion and acceleration/deceleration. The corresponding cost matrix is shown in Fig. 4.4b.

The initial optimal assignment is $\mathbf{X}_0^* = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$. If global communication and computation are reliable and not prohibitively expensive, the robots may use SA directly. If not, the central unit computes the maximum cost difference (ii) to decide whether the robots respond to changes. The worst cost sum when the robots keep the initial assignment is 100, and the best cost sum when they consider cost changes is 50 (i.e., when $\mathbf{X}_1^* = \left(\begin{smallmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{smallmatrix}\right)$. If the central unit decides that the difference (loss) 50 is too large, it tries to find cliques (i) but there is no such clique in this example. Therefore, the robots finally cope with cost changes in an incremental fashion (iii) from local communication.

4.2.2.2 Computing $\theta(\mathbf{X}^*)$

A feasible solution must include n optimal variables (correspond to optimal assignments). Among the remaining $n^2 - n$ variables, $n - 1$ variables need to be chosen to complete a feasible solution which consists of $2n - 1$ basic variables. Our implementation SA($\mathbf{X}_0^*, \mathbf{C}_0$) enumerates all $|k| = \binom{n^2-n}{n-1}$ feasible solutions to compute (2.7). The running time grows factorially as the input size increases. However, the interiors of R_k may overlap and $\theta(\mathbf{X}^*)$ could be covered by a subset of R_k. A method such as [40] can be used to find nonoverlapping subsets of $\theta(\mathbf{X}^*)$ which requires less effort than enumerating all feasible solution sets.

We develop a randomized anytime algorithm RandSA($\mathbf{X}_0^*, \mathbf{C}_0$) to facilitate a faster computation of $\theta(\mathbf{X}^*)$. A partial enumeration of degenerate solutions bring an incomplete set, which is $\theta'(\mathbf{X}^*)$, but the incomplete set often takes a large portion of $\theta(\mathbf{X}^*)$. From this observation, we implement a randomized anytime algorithm that randomly chooses degenerate solutions without replacement and computes corresponding critical regions (2.6) as much as possible with given time. The random sampling is done without replacement so $\theta'(\mathbf{X}^*)$ is nondecreasing during an execution. We prove that the randomized method is complete so eventually computes an exact solution.

Theorem 4.9 The randomized anytime sensitivity analysis is complete.

Proof. The full set of linear inequalities $\theta(\mathbf{X}^*)$ is the union of R_k where $k \in H$, as defined by (2.7). For each k, R_k could be a complete set or a partial set of $\theta(\mathbf{X}^*)$. The randomized anytime method randomly chooses a feasible solution k and does not replace the sample. If R_k is a complete set, the algorithm finds the complete $\theta(\mathbf{X}^*)$ and accordingly finds the exact interval of α. In the worst case, the algorithm chooses all samples, so all feasible solutions are picked to compute all R_k for $k \in H$. Therefore, the union of R_k forms the complete $\theta(\mathbf{X}^*)$. $\square$

The randomized method can be modified to sample with replacement, which has the advantage that the method needs less memory space since it does not keep track of the sampled R_k's have been seen. Doing so sacrifices completeness, though it may converge to quickly. It is also straightforward to bound the likelihood of providing a complete set because there are $\binom{n^2-n}{n-1}$ choices.

4.2.2.3 Factorizing a team of robots

Factorization can be done by analyzing all possible assignments $\mathbf{X}_q^*$ for $q = 0, \cdots, N-1$ computed by Alg. 4. An assignment problem has at least 4-dimension (two robots and two tasks) so it is hard to visualize geometry of the problem. Thus, we describe a figurative 2-D representation in Fig. 4.6a. One difference of the 2-D representation from higher dimensional cases is that all linear equations in $\theta(\mathbf{X}^*)$ are greater or equal to zero (see (2.6)), but the upper boundary of $\theta(\mathbf{X}^*)$ in Fig. 4.6a has the opposite inequality.

We have an initial optimal assignment $\mathbf{X}_0^*$ and its $\theta(\mathbf{X}_0^*)$ (Alg. 4, line 2-3). l is an arbitrary linear boundary in $\theta(\mathbf{X}^*)$. If the objective value is greater or equal to zero when l is maximized over the shaded area[4] (line 6), l contains the entire cost set (the shaded

[4]All l should be maximized because of the inequalities of them (see (2.6)).

area $\mathbf{C}$ in Fig. 4.6a). Otherwise, the shaded area is not covered by l (see Fig. 4.6b) thus a cost on l is perturbed to find a new $\theta(\mathbf{X}^*)$ that includes the rest of $\mathbf{C}$ (line 12-22). Once the current $\theta(\mathbf{X}^*)$ is expanded by perturbing points (i.e., costs), newly found $\theta(\mathbf{X}_q^*)$ are merged and checked also. The algorithm terminates if $\theta(\mathbf{X}^*)$ completely includes $\mathbf{C}$. It returns all $\mathbf{X}_q^*$ and $\theta(\mathbf{X}_q^*)$ found.

In Fig. 4.6a, the direction of a perturbation is toward $\mathbf{c}'$. The magnitude ϵ of the perturbation should be carefully chosen because a large ϵ may skip some $\theta(\mathbf{X}_q^*)$ on the way. We describe how we determine the optimal magnitude and the direction of a perturbation.

Theorem 4.10 Let l be an arbitrary linear boundary in $\theta(\mathbf{X}^*)$. The magnitude of a perturbation ϵ to perturb an arbitrary point $\mathbf{p}$ on l of $\frac{|\mathbf{p}|}{n}$ will not miss any $\theta(\mathbf{X}^*)$.

Proof. Let $\mathbf{l}$ be the normal of an arbitrary linear boundary l in $\theta(\mathbf{X}^*)$. From line 6 in Alg. 4, we have $\mathbf{c}'$, which is an extreme point of $\mathbf{C}$ outside of the current $\theta(\mathbf{X}^*)$. The projection of $\mathbf{c}'$ onto $\mathbf{l}$ is

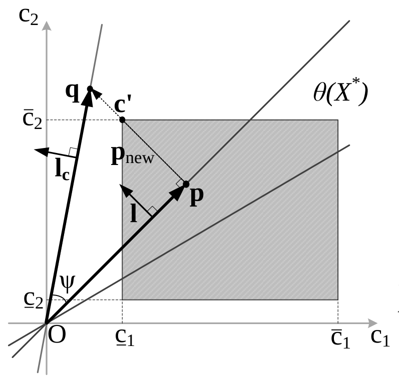
$$\mathbf{p} = \frac{\mathbf{c}' \cdot \mathbf{l}}{|\mathbf{l}|^2}\mathbf{l}.$$

Suppose that $\mathbf{l}_c$ is the normal vector of the nearest boundary to l (other than itself). Let $\mathbf{q}$ be a vector orthogonal to $\mathbf{l}_c$. The direction of movement from $\mathbf{p}$ to $\mathbf{q}$ is along a vector

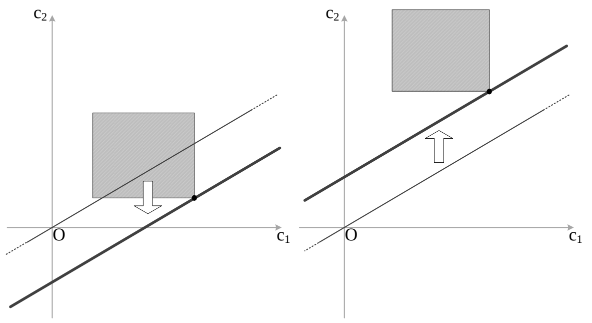
$$\mathbf{p}_{new} = \mathbf{p} + d(\mathbf{q} - \mathbf{p})$$

where $d = |\mathbf{p}| \tan\psi$ is the magnitude of the move. We look for the minimum d because $\mathbf{p}_{new}$ is toward the closest boundary. $\tan\psi$ is an increasing function in $(-\frac{\pi}{2}, \frac{\pi}{2})$. Therefore, if ψ is minimized, d is also minimized.

Normalized vectors of the above vectors are denoted as $\hat{\mathbf{l}}, \hat{\mathbf{l}_c}, \hat{\mathbf{p}}, \hat{\mathbf{p}_{new}}$, and $\hat{\mathbf{q}}$. Since $\hat{\mathbf{l}} \perp \hat{\mathbf{p}}$ and $\hat{\mathbf{l}}_c \perp \hat{\mathbf{q}}$, the angle between $\hat{\mathbf{l}}$ and $\hat{\mathbf{l}}_c$ is ψ as well. Therefore, $\hat{\mathbf{l}}_c \cdot \hat{\mathbf{l}}_c = \cos\psi$. Since $\psi = \arccos \hat{\mathbf{l}}_c \cdot \hat{\mathbf{l}}_c$, ψ is minimum at maximum $\hat{\mathbf{l}}_c \cdot \hat{\mathbf{l}}_c$.



(a) A 2-D figurative picture.



(b) Objective values of two cases.

Figure 4.6: A 2-D figurative representation of cost space. (a) Bold lines represent linear boundaries (hyperplanes) of $\theta(\mathbf{X}^*)$ and the shaded area represents a cost matrix $\mathbf{C}$ bounded by $\underline{\mathbf{C}}$ and $\bar{\mathbf{C}}$. (b) If a boundary does not cover all shaded area, the objective value of maximization over the area is negative (left). Otherwise, the value is nonnegative (right).

As we discussed in Section 2.2.2, coefficients (normals) of linear boundaries in $\theta(\mathbf{X}^*)$ are $-1, 0$, or 1. To make a dot product of two normals of boundaries maximum, the boundaries should have the maximum number of 1's (or the maximum number of -1's). $\hat{\mathbf{1}}$ is the case but two boundaries should be distinct. Therefore, the product of $\frac{1}{\sqrt{n^2}}[1\,1\,1\cdots1\,1]$ and $\frac{1}{\sqrt{n^2-1}}[1\,1\,1\cdots1\,0]$ is the maximum, that is $\frac{n^2-1}{\sqrt{n^2}\sqrt{n^2-1}}$ where n is the dimension of the cost space.

Now we have $\psi_{min} = \arccos\frac{n^2-1}{\sqrt{n^2}\sqrt{n^2-1}}$. Therefore,

$$d_{min} = |\mathbf{p}|\tan(\arccos\frac{n^2-1}{\sqrt{n^2}\sqrt{n^2-1}}) = |\mathbf{p}|\frac{1}{\sqrt{n^2-1}},$$

which is the distance to the closest boundary along $\mathbf{p}_{new}$. We set a safe magnitude of perturbation $\epsilon = \frac{|\mathbf{p}|}{n}$, which does not skip any $\theta(\mathbf{X}^*)$. $\square$

Here we observe that only line 6 in Alg. 4 is related to the shape of cost boundary. In line 6, a cost boundary is a feasible region of the linear programming problem. If the feasible region is linearly constrained convex (i.e., a hyper-polytope), on may still find the

Algorithm 4 FindTheta

Input: An $n \times n$ cost matrix $\mathbf{C}_0$, $\underline{\mathbf{C}}$, and $\bar{\mathbf{C}}$
Output: A set of assignments $\mathbf{X}^*_q$ and $\theta(\mathbf{X}^*_q)$ for $q = 0, \cdots, N-1$

```
1  $i = 0, q = 1$
2  $\mathbf{X}^*_0$ = Hungarian$(\mathbf{C}_0)$
3  $\theta_0(\mathbf{X}^*)$ = SA$(\mathbf{X}^*_0, \mathbf{C}_0)$ // compute (2.7)
4  $\theta(\mathbf{X}^*) = \theta_0(\mathbf{X}^*)$
5  while (1)
6     $(\mathbf{c}'_i, obj_i)$ = linprog$(\mathbf{l}_i, \underline{\mathbf{C}}, \bar{\mathbf{C}}, \max)$ // max $\mathbf{l}_i$ over the bounds
             // $\mathbf{l}_i$: the normal of the $i^{\text{th}}$ linear boundary $l_i$ in $\theta(\mathbf{X}^*)$
7     if $obj_i \geq 0$ // if $\mathbf{C}$ does not satisfy $l_i \geq 0$
8        $i = i + 1$
9        if $i = |\theta(\mathbf{X}^*)|$ // if all linear boundaries in $\theta(\mathbf{X}^*)$ are checked
10          break
11       end if
12    else // perturb a point $p$ on $l_i$ toward $\mathbf{c}'$ to find
          // a new $\mathbf{X}^*$ and $\theta(\mathbf{X}^*)$
13       $\mathbf{p} = \frac{\mathbf{c}' \cdot \mathbf{l}_i}{|\mathbf{l}_i|^2}\mathbf{l}_i$ // $\mathbf{p}$ is a projection of $\mathbf{X}_i$ onto $l_i$
14       $\epsilon = \frac{|\mathbf{p}|}{n}$
15       $\mathbf{p}_{\text{new}} = \mathbf{p} + \epsilon(\mathbf{c}' - \mathbf{p})$
16       $\mathbf{C}_q$ = reshape$(\mathbf{p}_{\text{new}}, n)$ // reshape a vector to an $n \times n$ matrix
17       $\mathbf{X}^*_q$ = Hungarian$(\mathbf{C}_q)$
18       $\theta(\mathbf{X}^*_q)$ = SA$(\mathbf{X}^*_q, \mathbf{C}_q)$
19       $\theta(\mathbf{X}^*) = \theta(\mathbf{X}^*) \cup \theta(\mathbf{X}^*_q)$
20       $i = 0$
21       $q = q + 1$
22    end if
23 end while
24 return $\{\mathbf{X}^*_0, \cdots, \mathbf{X}^*_{N-1}\}$ and $\{\theta(\mathbf{X}^*_0), \cdots, \theta(\mathbf{X}^*_{N-1})\}$
```

vertices of the cost boundary with the maximum objective value by linear programming; this preserves the same time complexity. If the feasible region is convex but nonlinear, we can use convex optimization methods such as the interior point method, and the problem is still in P as shown in Appendix B. However, the problem becomes NP-hard if the feasible region is nonconvex (either linear or nonlinear) [7].[5]

These preceding hardness results show that the analysis of the assignment optimality can include interrelated costs essentially for free if they are linear, but are tractable even for costs related to problems like that of Zermelo's. Furthermore, experimental results in Section 4.3 show that treating interrelated costs as independent significantly decreases the value of the analysis.

Cliques can be found easily by summing the assignments found that is $\mathbf{X}_C = \sum_{q=0}^{N-1} \mathbf{X}_q^*$. If there are elements with zero in $\mathbf{X}_C$, the robot-task pairs are never assigned. If a block diagonal matrix can be found, the main diagonal blocks represent cliques. With local communication and computation within each clique only, the robots achieve global optimality.

4.2.2.4 Choosing to persist with the initial assignment

The pseudocode is shown in Alg. 5. All assignments $\mathbf{X}_q^*$ and $\theta(\mathbf{X}_q^*)$ for $q = 0, \cdots, N-1$ are given by Alg. 4 (line 1). For each $\theta(\mathbf{X}_q^*)$, find the minimum costs $\mathbf{C}_q$ over $\theta(\mathbf{X}_q^*)$ with the bounds $\underline{\mathbf{C}}$, and $\bar{\mathbf{C}}$ (line 3). For each q, we have an assignment $\mathbf{X}_q^*$ and the minimum cost $\mathbf{C}_{\min_q}$. In line 6, $\min \mathbf{C}_{\min_q} \mathbf{X}_q^*$ returns the minimum cost sum of N assignments. On the other hand, we can compute the maximum cost sum if robots do not change their initial assignment, by computing $\bar{\mathbf{C}}\mathbf{X}_0^*$. Therefore, line 6 gives the maximum cost loss when robots persist the initial assignment while having no communication and re-assignment.

One can decide with c_{worst} whether to persist with the initial assignment by considering

[5]There are polynomial-time approximation algorithms for such nonconvex optimization problem (see the review of optimization solvers in [69]). Therefore, we can replace the linear programming solver (line 6 in Alg. 4) by a solver appropriate to the shape of the feasible region.

Algorithm 5 MaxLoss

Input: An $n \times n$ cost matrix $\mathbf{C}_0$, $\underline{\mathbf{C}}$, and $\bar{\mathbf{C}}$

Output: A maximum cost difference c_{worst}

```
1 $(\{\mathbf{X}_0^*, \cdots, \mathbf{X}_{N-1}^*\}, \{\theta(\mathbf{X}_0^*), \cdots, \theta(\mathbf{X}_{N-1}^*)\}) = \text{FindTheta}(C_0, \underline{\mathbf{C}}, \bar{\mathbf{C}})$
2 for $q = 0$ to $N - 1$
3     $(\mathbf{c}'_q, obj_q) = \text{linprog}(\mathbf{c}, \theta(\mathbf{X}_q^*), \underline{\mathbf{C}}, \bar{\mathbf{C}}, \min)$
                          //minimize costs over $\theta(\mathbf{X}_q^*)$ with the bounds
4     $\mathbf{C}_q = \text{reshape}(\mathbf{c}'_q, n)$
5 end if
6 $c_{\text{worst}} = \max\{\bar{\mathbf{C}} \odot \mathbf{X}_0^* - \min(\mathbf{C}_{\min_q} \odot \mathbf{X}_q^*)\}$
7 return $c_{\text{worst}}$
```

the computational/communication expense of a re-assignment.

4.2.2.5 Incremental communication

$\theta(\mathbf{X}^*)$ can be summarized in different ways to show relevant information about the effect of cost changes. A one-dimensional cut yields a lower and an upper bound for each cost. This interval is valid if all other costs remain unchanged. But α-dimensional cuts allow simultaneous changes of the α costs, but $n^2 - \alpha$ costs must remain unchanged. The tolerance approach finds the maximum tolerance percentage of the costs that finally gives a tolerance region. The region is a hyper-cuboid in which each dimension is bounded by an interval $c_{ij} - \tau_{ij} \leq c_{ij} \leq c_{ij} + \tau_{ij}$ where $\tau_{ij} \in \mathbb{R}^{\geq 0}$. (See [104] and the more recent advance by Filippi [36] for details.)

This intervals (call this τ-interval for distinction) is not larger than the 1-D cuts of $\theta(\mathbf{X}^*)$, but they are independent from other cost changes. It is attractive in multi-robot systems because robots do not need any communication for cost changes, unless their own intervals are violated by cost changes. On the other hand, even though one of a robot's costs violates its τ-interval, other cost changes countervail the violation. For example, an increase of a cost violates the interval, but a decrease of another cost could retain the optimal assignment. We develop an algorithm shown in Alg. 6 that incrementally checks a violation from a robot itself to adjacent robots.

$\theta(\mathbf{X}_0^*)$ and τ-intervals of the initial assignment are computed and distributed to robots (line 2-5). Then the following procedure runs on each robot R_i concurrently. If c_{ij} violates its τ-interval, the costs are collected in a set C_{v_i} (line 6-11). R_i checks $c_{ij} \in C_{v_i}$ altogether whether they satisfy $\theta(\mathbf{X}_0^*)$. (Use c_{ij} of $\mathbf{C}_0$ for $c_{ij} \notin C_{v_i}$.[6]) The checking returns $V_i \in \{0, 1\}$ where $V_i = 0$ means that the cost changes turn out not to violate $\theta(\mathbf{X}_0^*)$ and otherwise $V_i = 1$. It can be done simply by substituting cost variables in $\theta(\mathbf{X}_0^*)$ by the initial and changed costs (line 13). If $V_i = 0$, the algorithm terminates and return V_i to the central unit. Otherwise, R_i finds an adjacent robot and receives its changed cost set C_{v_a}.[7] If any of R_i finally returns $V_i = 1$, the robots need global communication; if none of R_i returns $V_i = 1$, their cost changes do not alter the current assignment, and this fact is checked without global communication.

4.2.2.6 Complexity analysis

Computing $\theta(\mathbf{X}^*)$ has $O(|k|n^2)$ time complexity where $|k|$ is the number of degenerate solution sets. Each k has $(n-1)^2$ linear boundaries so there are at most $|k|(n-1)^2$ boundaries. Alg. 4 is dominated by $\theta(\mathbf{X}^*)$ computation in the while loop (lines 5–23). Each loop iterates if a new $\theta(\mathbf{X}^*)$ has found. Therefore, the time complexity is $O((|k|n^2)^N)$ where N is the number of possible assignments with cost matrix $\mathbf{C}$. Alg. 5 executes Alg. 4 first, and an $O(n^3)$ LP runs N times. Then it has $O((|k|n^2)^N + Nn^3) = O((|k|n^2)^N)$. Alg. 6 includes SA so is dominated by it, but the remainder of the procedure, which runs on each robot has $O(n)$ time complexity (we ignoring the costs of inter-robot communication in this analysis). If Alg. 5 follows after executing Alg. 4, its complexity is $O(Nn^3)$ since it

[6]This needs an assumption that $c_{ij} \notin C_{v_i}$ remain unchanged.

[7]We assume there is at least one robot in range. If there is no such robot, R_i can navigate or wait to have a robot.

Algorithm 6 IncrementalComm

Input: An $n \times n$ cost matrix $\mathbf{C}_0$, $\underline{\mathbf{C}}$, and $\bar{\mathbf{C}}$
Output: Indicator variables $\{V_1, \cdots, V_n\}$

```
1  $l = 1, V_1, \cdots, V_n = 0, C_{v_i} = \emptyset$
2  $\mathbf{X}_0^* = \text{Hungarian}(\mathbf{C}_0)$
3  $\theta(\mathbf{X}_0^*) = \text{SA}(\mathbf{X}_0^*, \mathbf{C}_0)$
4  $\mathcal{T} = \text{TA}(\theta(\mathbf{X}_0^*), \mathbf{C}_0)$ // compute $\tau$-intervals:  $\mathcal{T}_{ij} = [c_{ij} - \tau_{ij}, c_{ij} + \tau_{ij}]$
5  Distribute $\theta(\mathbf{X}_0^*)$ and $\mathcal{T}_{ij}$ to corresponding $R_i$
                    // Below lines run on each robot $R_i$ concurrently
6  for $j = 1$ to $n$ // $i$ is fixed to each robot's index
7    if $\underline{c}_{ij} < c_{ij} - \tau_{ij}$ and $c_{ij} + \tau_{ij} > \bar{c}_{ij}$ // if $\mathcal{T}_{ij}$ is violated,
8      $V_i = 1$ // there is at least one violation in $R_i$'s cost
9      $C_{v_i} = C_{v_i} \cup c_{ij}$ // collect violated costs
10   end if
11 end for
12 while $|C_{v_i}| \leq n^2$ // while not all costs are included
13   $V_i = \text{Check}(\theta(\mathbf{X}_0^*), C_{v_i}, \mathbf{C}_0)$ // check $C_{v_i}$ altogether
14   if $V_i = 0$
15     break
16   end if
17   $(R_a, C_{v_a}) = \text{FindAdjacent}(R_i)$ // $R_a$ is an adjacent robot
18   $C_{v_i} = C_{v_i} \cup C_{v_a}$
19 end while
20 return $V_i$ // $V_i = 1$ if global comm. needed, otherwise $V_i = 0$
```

uses the output of Alg. 4.

The worst-case time complexity is not polynomial to input size because N and $|k|$ are on the order of a factorial of n. However, not all $c_{ij} \in \mathbb{R}^{\geq 0}$ are likely to be considered because it is a bounded region, and the number of possible assignments is manageable. Also, $|k|$ can be reduced by using known methods such as that in [40], as discussed.

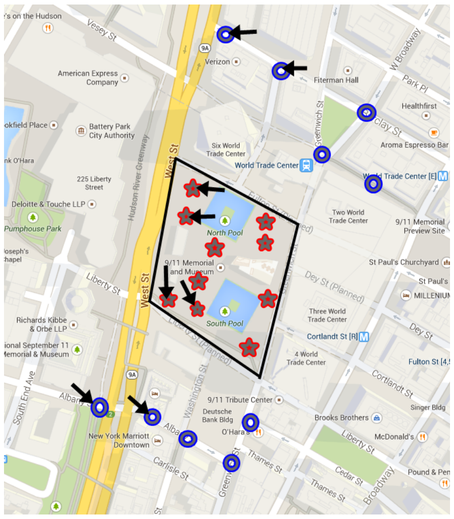
4.3 Experiments

We consider two scenarios based on reality where cost is traveling time. Both employ the same assumptions as the example in Section 4.2.2.1. The first one is a rescue scenario shown in Fig. 4.7a. Here, 10 victims (red stars) are inside a disaster site (black polygon) and 10 robots (blue circles) are outside. The robots navigate into the site while pushing debris. The robots move with 1 m/s speed and meet debris at every 10 m. The time to push one object is $t_w \in [0, 1]$. The second scenario is the multi-robot navigation problem shown in Fig. 4.7b, where 30 Robots and 30 tasks are uniformly distributed in a bounded area. The robots move at 10 m/s and encounter a traffic signal at every 300 m. The waiting time for the signal is $t_w \in [0, 30]$. Distances from the robots to the tasks are collected using the Google API [48]. The raw data are in meters but converted to time (sec) by considering robots' moving speeds.

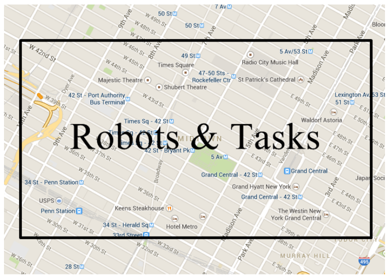
4.3.1 Computing $\theta(\mathbf{X}^*)$

The running times of the exact method are 0.014, 0.0764, 1.4524, 98.0622 sec for $n = 3, 4, 5, 6$, respectively (variances are 0.001, 0.006, 0.0021, 0.1947, and 10 iterations). Since the number of degenerate solutions $|k|$ has factorial growth as n increases, the running time is not fast for larger problem instances. However, as discussed above, $|k|$ can be reduced. With large sizes, the anytime algorithm we suggested can help decrease running-time.

As discussed in Section 4.2.2.2, the randomized anytime algorithm computes a partial



(a) A rescue scenario. Stars are victims and circles are robots.



(b) A navigation scenario. Randomly distributed robots and tasks.

Figure 4.7: Experimental setup for the multi-robot navigation problem. The marked robots and tasks in (a) are specially chosen for Section 4.3.3.

set $\theta'(\mathbf{X}^*)$. Fig. 4.8 shows percentages of the area from the randomized method with respect to the complete $\theta(\mathbf{X}^*)$ from the exact method when $n = 4$. For each time step, the percentage is measured by 100 uniform samples over $c_{ij} \in [0, 1000], \forall i, j$. The randomized method quickly approaches to 100% (at 0.0226 sec, it is same with the exact method). This means that $\theta(\mathbf{X}^*)$ from this randomized method is likely good enough to useful in most practical instances. Since the topic of this chapter is not about improving running time *per se*, we use the exact method in this section to find a theoretically complete region.

4.3.2 Reducing futile effort

We compare systems with the Hungarian method, 1-D intervals, and SA to see how efficiently they deal with cost changes. Suppose that there are multiple consecutive updates to costs given to the central unit. A system using the standard Hungarian method must execute the algorithm at every update to ascertain whether the updated costs alter the current assignment. Some re-computations find new assignments, but the others would be fruit-

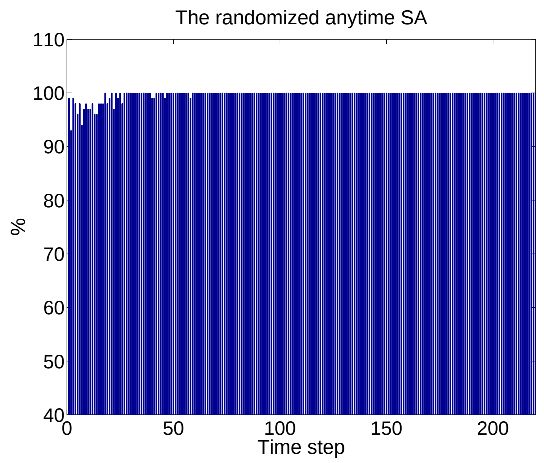


Figure 4.8: The performance of the randomized anytime algorithm. It quickly approaches to 100% which is the result from the exact method.

less attempts. Employing the 1-D interval method (e.g., iHM) saves some re-computation, attempting a new assignment when any of the intervals are violated. Nevertheless some re-computation would still be in vain because the method fails to consider simultaneous cost changes. Lastly, a system with SA does not recompute an assignment unless changed costs actually alter the current assignment. We measure the number of effective re-computations with the same cost changes. We compare the standard Hungarian method and the 1-D cuts of $\theta(\mathbf{X}^*)$, which are identical to the intervals from the iHM, and $\theta(\mathbf{X}^*)$.

Given an arbitrary $n \times n$ cost matrix (for $n = 3, 4, 5$), an optimal assignment was computed. Then 50 random matrices, uniformly sampled between $[0, 2]$, are added to the cost matrix. The result is shown in Fig. 6.6 and Table 6.1. The success rate is computed by $(\#$ of assignment changes$/\#$ of re-computations$) \times 100$. The result clearly shows that SA reduces unnecessary computations and communication.

4.3.3 Factorizing a team of robots

For each scenario, we randomly choose four robots and four tasks from the data collected. For each chosen problem instance, we run Alg. 4. The result is shown in Fig. 4.10

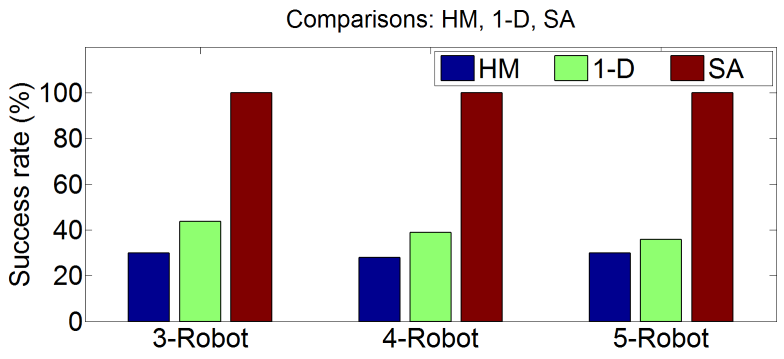


Figure 4.9: Comparisons of different approaches with respect to cost changes.

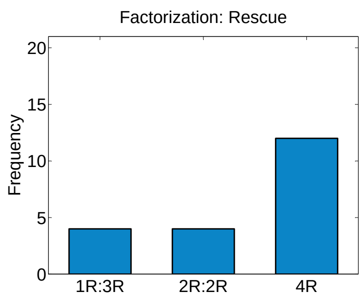
Table 4.1: Comparisons of different approaches with respect to cost changes. (The Hungarian method, 1-D intervals, sensitivity analysis.)

	$n = 3$			$n = 4$			$n = 5$		
Method	Attempts	Success	Rate	Attempts	Success	Rate	Attempts	Success	Rate
HM	50	15	30.00%	50	14	28.00%	50	15	30.00%
1-D	36	15	43.59%	36	14	38.88%	42	15	35.71%
SA	15	15	100%	14	14	100%	15	15	100%

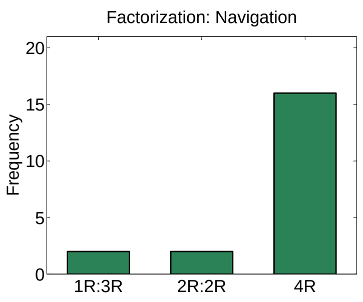
and Table 4.2 (2R:2R means that there are two cliques of two robots). Even though the scenarios do not have obvious spatial sparsity and/or locality, the algorithm is able to detect cliques when a team has such structure. The average running times of two scenarios are 2.0210 sec and 2.2768 sec ($\sigma^2 = 0.1144$, $\sigma^2 = 0.1851$ for 20 iterations), respectively. We also report results of factorization when tasks do have strong spatial locality (Fig. 4.10c). Two robots and two tasks are located as the marked robots and tasks in Fig. 4.10.

Table 4.2: Factorization results. Frequencies of cliques found (20 iterations).

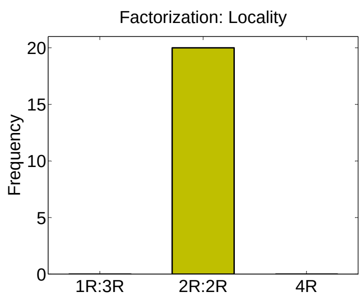
Clique size	Frequency		
	Rescue	Navigation	Locality
1R:3R	4	2	0
2R:2R	4	2	20
4R only	12	16	0



(a) The rescue scenario.



(b) The navigation scenario.



(c) Tasks with spacial locality.

Figure 4.10: Factorization results. Frequencies of cliques found (20 iterations).

Table 4.3: The maximum loss of persisting assignment. Some examples of execution results are shown (navigation scenario). The middle three columns shows cost sums (sec), and the last column shows running time (sec).

Size	Persist	Change	Max Loss	Time
2R	867.4	591.0	276.40	0.0624
3R	1036.6	518.2	518.4	0.8424
4R	1885.2	895.7	989.5	16.1305

4.3.4 Persisting with an initial assignment

Table 4.3 shows examples of maximum cost losses for different sizes of team. One (the central unit or an operator) can decide whether to execute the initial assignment without having any communication and computation using the cost loss information. If the communication/computation expenses are prohibitive, it would be beneficial to persist the initial assignment.

4.3.5 Incremental communication

Finally, we show how few communication messages are actually needed to detect whether optimality has been violated by cost changes. For each scenario, we compute the τ-intervals and distribute them to the robots. Each robot independently performs its task unless its costs violate the τ-intervals. Once any robot has intervals violated, the

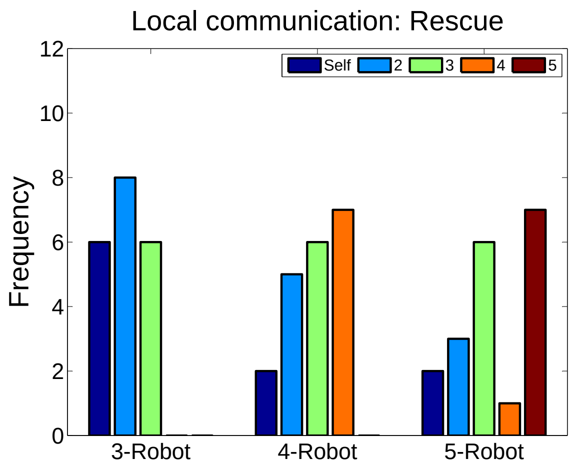
Table 4.4: Frequency of communication ranges. The bold numbers indicate the frequencies of local communications. For example, in the rescue scenario, 3-robot team has 6 self checks and 8 two-robot communications.

Range / Team size	Rescue							Navigation						
	Self	2	3	4	5	Time (sec)		Self	2	3	4	5	Time (sec)	
						Mean	Var						Mean	Var
3R	**6**	**8**	6	N/A	N/A	0.0769	0.0004	**11**	**5**	4	N/A	N/A	0.0476	0.002
4R	**2**	**5**	**6**	7	N/A	0.5156	0.0471	**7**	**5**	**1**	7	N/A	0.4898	0.373
5R	**2**	**3**	**6**	**1**	7	20.0539	55.4456	**1**	**6**	**2**	**1**	10	19.5001	60.9115

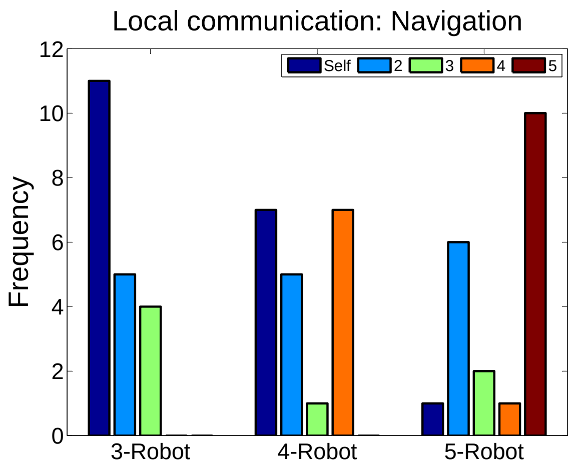
robot runs the individual procedure in Alg. 6. For each changed set of costs, we check how many robots are involved in communicating, and record the frequency of occurrence for this number. A team may have several local checks, but one robot may require global communication. In such a case, we record the largest communication needed among the robots. Note that we ensure every robot has at least one violation so all robots execute Alg. 6. We randomly choose robots and tasks from the data sets. We change the team size from three to five. Fig. 4.11 shows the results (for 20 iterations). In many executions, purely local communication is enough (bold numbers in Table 4.4) to see how the costs changes affect optimality of the current assignment. Note that running time includes the central unit's computation time for the τ-interval and $\theta(\mathbf{X}^*)$. As the team size increases, the running time increases as there is a combinatorial number of local communications. The variance also increases because an early termination takes very short time while additional local communications take an amount of time related to a combinatorial factor.

4.3.6 Interrelated costs

We assume that costs have linear relationships and, like Fig. 4.3a, the robots share one resource to perform tasks. Thus, any vicissitude affecting the resource changes all costs by the same amount. For example, if there is a delay on a common route to reach tasks, all traveling times increase accordingly by the same amount. We randomly choose an $n \times n$ cost matrix and a scalar value of delay from $\mathcal{U}(0, 10)$. We model the relationships of costs



(a) The rescue scenario.

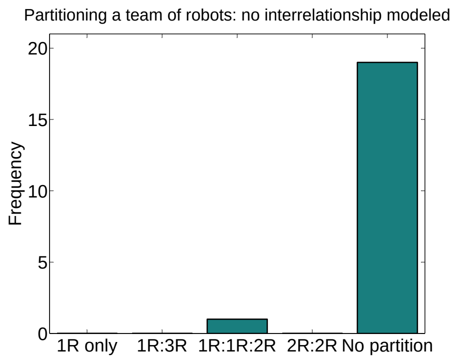


(b) The navigation scenario.

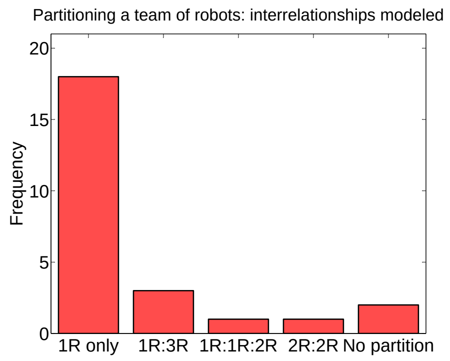
Figure 4.11: Frequency of communication ranges. For each team size, the left most bar means individual check whereas the right most bar mean global communication. Local communication is more frequent with Alg. 6.

through a set of linear equations (e.g., $c_{ij} - c_{pq} = t_{ij} - t_{pq}$ for $ij \neq pq$ where t is a nominal cost without delay). We run Alg. 4 to compute all optimal assignments under the cost boundary modeling interrelationships of costs. We also run the algorithm with a simple bounded region that does not model the interrelationships.

The output of the algorithm is all optimal assignments that a team of robots may have given a region of costs. In other words, assignments other than the output assignments never happen to the team. Thus, a small number of output assignments means that there are only few scenarios in task assignment. The results show that the number of output assignments is greatly reduced when interrelationships are modeled (averagely from 9.95 to 1.85 as shown in Table 4.5a). From this analysis, we are able to find partitions in a team of robots in which each partition can work independently without communicating with other robots outside. Specifically, partitions can be found by summing the assignment matrices found. If there are elements with zero in the summed matrix, the corresponding robot-task pairs will be never assigned. If a block diagonal matrix can be found, the main



(a) No interrelationship modeled.



(b) Interrelationships modeled.

Figure 4.12: The results of partitioning a team of robots. Modeling interrelationships reduces false positives in computing all possible assignments within a cost region. Thus, more partitions can be found with the smaller number of assignments.

diagonal blocks represent cliques. The results are shown in Fig. 4.12 and Table 4.5(b). The cost region that models interrelationships removes false positives of possible optimal assignments, so more partitions can be found.

Table 4.5: The results from sensitivity analysis for cost regions.

(a) The number of assignments from given cost regions (20 iterations).

Interrelationship	Not modeled	Modeled
Mean	9.9500	1.8500
Std. dev.	4.2855	1.0984

(b) The frequency of partitions found when $n = 4$ (20 iterations).

Partition type	Interrelationship	
	Not modeled	Modeled
1R's	0	18
1R:3R	0	3
1R:1R:2P	1	1
2R:2R	0	1
No partition	19	2

4.4 Summary

In this chapter, we propose a cost representation that incorporates uncertainty in costs and models interrelated costs. The representation assumes that costs are bounded by a finite region. We employ a sensitivity analysis approach for multi-robot task allocation and compare it with other methods, showing that is advantageous when costs change. We also proposed three methods that reduce centralization of multi-robot systems alongside the basic routine for computing $\theta(\mathbf{X}^*)$ and a fast approximate version, which is a randomized anytime algorithm. We examined our algorithms with realistic scenarios and data, not merely randomly generated matrices. We also show that modeling interrelationships yields tighter cost regions and hence better predictions for how the optimality of the task allocation under interrelated and uncertain costs.

5. ANALYZING THE SENSITIVITY OF THE OPTIMAL ASSIGNMENT IN PROBABILISTIC MULTI-ROBOT TASK ALLOCATION

In MRTA, the most common formulation involves, firstly, a cost being estimated for each robot–task pairing, and then the computation of an assignment that minimizes the sum of costs, i.e., the ST-SR-IA MRTA problem. When circumstances change or new information comes to light (e.g., such as the discovery of a new task, deployment of additional robots, or the unexpected occurrence of an event) the assignment of robots to tasks may need to be adjusted to reflect these contingencies. By doing so repeatedly and with regularity, the robots can produce dynamic cooperative behavior that befits a team.

The literature on MRTA is expansive but in almost all treatments the estimated costs are scalar values, failing to capture any uncertainty in the states of the robots, or the tasks, or the environment. A further, and also nearly universal, assumption of the ST-SR problems is that the costs are independent, having no interrelationship between values. Practical inefficiencies can result rather easily from ignoring either such interdependencies or uncertainty. Even very straightforward scenarios lead to MRTA problem instances where these assumptions, though standard, are dubious. This chapter investigates the problem of optimal assignments when neither of these simplifying assumptions are made.

This chapter explores a way to capture specific forms of uncertainty and to incorporate interrelationships between costs because we are interested, generally, in richer cost representations along both dimensions. We consider costs as random variables for which distributional information is available. Such the information could be obtained from robots employing a state estimator or historical measurements from a dataset. For example, consider the navigation example in Fig. 5.1 in which the robot and the task have position

This chapter, authored by Changjoo Nam and Dylan Shell, will appear in *IEEE Robotics and Automation Letters* as a regular article with the same title.

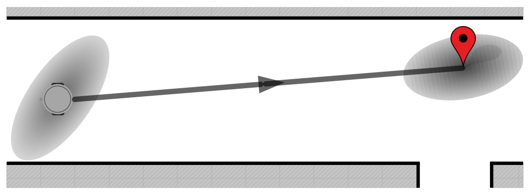


Figure 5.1: Cost uncertainty can arise in many ways. For example, owing to position uncertainty of the robot and the task. The sum of two independent normal distributions is also normally distributed, so the distance between a robot and a task is normally distributed. The traveling time is proportional to the distance, so time spent navigating (a useful metric of cost) is also normally distributed.

uncertainties represented by 2×2 covariance matrices output from a Kalman filter. Uncertainty in the robot's pose and its estimate of the task's position mean that traveling time is uncertain too. More specifically, the cost is normally distributed if positions also have normal distributions because the sum of two independent normal distributions is also normal. In this trivial example, we are assuming that no contingencies arise while the robots are traveling and that the cost (in time) is simply proportional to the distance.

In the present work, we characterize costs by their mean and Conditional Value-at-Risk (CVaR), the latter is a risk measure suitable for any type of distribution. The chapter describes statistical properties of optimal assignments given the characterizations of costs (i.e., the mean and CVaR) and a precise form of interrelationship in the costs that we can model tractably. We examine an efficient model for computation of optimal assignments subject to a risk preference that determines the relative importance between the mean and CVaR, when there is a tension between them. We show that there is a useful class of assignments that are indifferent to this risk preference. For problems outside of this class, we introduce a fast heuristic algorithm which computes the sensitivity of an optimal assignment to the particular risk preference.

5.1 Related Work

Assignment problems where costs are random variables are termed *random assignment problems*, an area of extensive research that was first surveyed comprehensively by Burkard and Cela [21] and, more recently, by Krokhmal and Pardalos [64]. Broadly speaking, *exact analysis* studies problems in terms of instance size, n, while *asymptotic analysis* considers $n \to \infty$. These analyses provide the upper and lower bounds, and the expected value of the cost sum of an optimal assignment. The values are expressed as functions of n, allowing one to understand behaviors of the problem according to its size easily (see [21] for more detail). These work has a wide range of applications in practical problems where costs have uncertainties. The result gives a useful information to system designing or decision-making process. In the probabilistic MRTA, we may use the analyses to see how a realization of the cost sum of an optimal assignment, which is determined based on only the expected values of the costs, is close to the best- or the worst-case. However, the assumptions of homogeneous distributions (e.g., mostly the uniform $[0, 1]$ distribution) and the independent and identically distributed random variables limit exploring more complex settings where costs are dependent and drawn from various distributional assumptions.

Nikolova and Stier-Moses [87] propose a traffic assignment model that incorporates uncertainty by introducing stochastic costs. There, a cost is the sum of an expected travel time and a random variable representing the uncertainty in the time. They assume that all random variables are uncorrelated. Each agent chooses a set of routes in a network and has an objective value that is the weighted sum of the expected travel time and the standard deviation of its travel time along the routes. Their work provides equilibria for several versions of routing games. This stochastic cost representation has been successfully used and analyzed in game theoretic perspective of the routing problem, and now we are interested in applying the stochastic representation and the risk-averse formulation to MRTA.

The closest work to this chapter is that of Ponda et al. [93], who propose a stochastic formulation of task allocation problems where planning parameters have uncertainties owing to the discrepancies between system models and actual system dynamics. The chance-constrained approach, which maximizes the worst-case cost sum within a risk threshold, is used. The proposed framework allows agents to work in a distributed manner by allocating individual risk threshold parameters based on a global risk threshold. The result shows that their planner outperforms the deterministic and the robust planners for any risk threshold. However, the consequence of a particular value of the global threshold cannot be anticipated directly in terms of the resulting allocation of agents to tasks. Thus, a necessity of analyzing the outcomes of those threshold values arises to consider countermeasures (e.g., reallocations) against the uncertainties.

5.2 Preliminary: Risk Measures

In financial mathematics, risk is defined as the variability of the future value of a position [5], and a risk measure maps a set of random variables into the set of real numbers. In the 1950's, Markowitz initiated the research of optimizing a portfolio considering the overall return and risk. In [76], variance is employed as the risk measure of a portfolio. However, the variance as a risk measure has been criticized as having the drawback that it treats the positive and negative deviations from the mean identically.[1] For this reason, other risk measures have been developed, such as Value-at-Risk (VaR) and Conditional Value-at-Risk (CVaR).

Given a confidence interval $\lambda \in (0, 1)$, VaR_λ is the smallest value such that the probability that a loss exceeds the value at most $(1 - \lambda)$. For example, if the VaR on an asset is \$100 at a confidence level 95%, the probability that a loss greater than or equal to \$100

[1]For example, the realizations of costs in the probabilistic MRTA less than the mean are actually beneficial.

is 0.05. CVaR_λ is the expectation of the worst $(1-\lambda)$ of the distribution. CVaR has the advantage over VaR because of the measure's convexity and sub-additivity [94]. Computations of CVaR are not simple (they involve integrals), but closed-form expressions of a large number of common distributions are provided in [75]. For those distributions that do not have closed-form expressions of CVaR, one may compute integrals or sample from the distribution and compute the average of the samples greater than or equal to VaR. If one has historical data, CVaR can be computed by fitting a density function to the data or by a nonparametric method by using the data directly. Interested readers are referred to Artzner et al. [5].

5.3 The Probabilistic Cost Representation

A straightforward way to model uncertainty is to treat the c_{ij} in (2.1) as random variables, C_{ij}. One may further express the observation that uncertainty of a cost arises from three sources: the robot, task, and the environment associated with the robot and task (e.g., the path between a robot and a task), via statistical properties that express interrelatedness in the costs. We consider costs of the form of a sum:

$$C_{ij} = \mathsf{R}_{ij} + \mathsf{T}_{ij} + \mathsf{E}_{ij}, \tag{5.1}$$

where the R_{ij}, T_{ij}, E_{ij} are random variables, representing costs contributed by the i^{th} robot, the j^{th} task, and an environment between them, subject to the following conditions:

1. every E_{ij} is independent of every R_{kl};
2. every E_{ij} is independent of every T_{kl};
3. every R_{ij} is independent of every T_{kl};
4. each E_{ij} is independent of $\forall k_{k\neq i}\, \forall l_{l\neq j}\ \mathsf{E}_{kl}$;
5. each R_{ij} is independent of $\forall k_{k\neq i}\, \forall l\ \mathsf{R}_{kl}$;

6. each T_{ij} is independent of $\forall k\, \forall l_{l \neq j}\ \mathsf{T}_{kl}$.

Note that there is no assumption of identical distributedness.

Statistical dependencies can exist between factors not precluded by the six conditions above. The term E_{ij} is a factor which influences C_{ij} independently of other factors. But the R_{ij} variable can have dependencies on R_{ik}, it is intended to capture uncertainty born of aspects of the i^{th} robot. The same thinking applies to T_{ij} for the j^{th} task.

Theorem 5.11 For costs of the form (5.1) subject to the six stated constraints, the costs C_{ij} where $x^*_{ij} = 1$ $(i, j \in \{1, \cdots, n\})$ are independent random variables given an assignment $\mathbf{X}^*$ satisfying (2.2)–(2.3).

Proof. Consider distinct C_{kl} and C_{rs}, where $x^*_{kl} = 1$, and $x^*_{rs} = 1$. Clearly E_{kl} is independent from E_{rs}, but R_{kl} can only be dependent on R_{rs} if $k = r$, which contradicts (2.3). Similarly, T_{kl} can only be dependent on T_{rs} if $l = s$, which contradicts (2.2). $\square$

The joint distribution of the costs must be known in order to compute the distribution of the sum of random costs. With the independence of the costs, the joint distribution can be obtained by the product of the given marginal cost distributions, which is significantly more convenient than that of the dependent case.

We prove another theorem about CVaR for a computationally tractable formulation of the probabilistic MRTA problem described in the following section. Definitions 5.12, 5.13, and Theorem 5.14 are from a comprehensive review of risk measures and comonotonicity [31].

Definition 5.12 (Comonotonic set) The set $A \subseteq \mathbb{R}^n$ is comonotonic if for any y and z in A, either $y \leq z$ or $z \leq y$ holds.

Definition 5.13 (Comonotonic random vector) A random vector is comonotonic if it has a comonotonic support.

Theorem 5.14 (Comonotonic additivity) In Theorem 4.2.1 in [31], it is proven that

$$\text{CVaR}_{\lambda,S^c} = \sum_{i=1}^{n} \text{CVaR}_{\lambda,Z_i}, \tag{5.2}$$

where $\text{CVaR}_{\lambda,V}$ denotes the CVaR of the random variable V with a confidence level λ. Here, S^c is the sum of comonotonic random variables $(Z_1^c, Z_2^c, \cdots, Z_n^c)$ such that $S^c = Z_1^c + Z_2^c + \cdots + Z_n^c$. Thus, the CVaR of the sum of the random variables is equal to the sum of the CVaRs of the random variables if the random variable forms a comonotonic random vector.

From the definitions and theorem above, we derive a theorem for a convenient computation of the CVaR of an assignment.

Theorem 5.15 The CVaR of an assignment $\mathbf{X}$ can be computed simply from the sum of each cost distribution's CVaR. Mathematically:

$$\text{CVaR}_{\lambda,C_{\mathbf{X}}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{CVaR}_{\lambda,C_{ij}} x_{ij}. \tag{5.3}$$

Proof. The cost sum of an assignment, $C_{\mathbf{X}}$, consists of the sum of costs (i.e., random variables) whose decisions variables are $x_{ij} = 1$ (i.e., $C_{\mathbf{X}} = \sum_{i,j=1}^{n} C_{ij} x_{ij}$). The support of those random variables is the complete set of, or a subset of, nonnegative real numbers, which is a totally ordered set. A totally ordered set is a comonotonic set by definition. So the sum of costs in an assignment is the sum of comonotonic variables, and Theorem 5.14 holds for the CVaRs of $C_{\mathbf{X}}$ and C_{ij}. $\square$

5.4 Optimal Assignment with Probabilistic Costs

For $i, j \in \{1, \cdots, n\}$, we use the mean μ_{ij} and $\text{CVaR}_{\lambda,ij}$ of random variable C_{ij} to characterize its distribution. By virtue of the additive property of CVaR in an assignment (as proved in Theorem 5.15), we can define two objective functions replacing (2.1), which are the sum of means and CVaRs:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \quad \mu_{ij} x_{ij}, \tag{5.4}$$

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \text{CVaR}_{\lambda,ij} x_{ij}. \tag{5.5}$$

It is worth noting that, without Theorem 5.11 and 5.15, one must compute the sum of the distributions and the CVaR of the sum, which involve several integrals. The problem of optimizing these functions subject to (2.2)–(2.5) is the biobjective assignment problem (BiAP) [102]. The optima are best visualized as a Pareto front. Some assignments may have low summed mean but high CVaR, some with low CVaR but high mean, and others may represent a compromise.

It is not meaningful to seek a single optimum given that there are two objectives unless we can express a preference as a precise trade-off between minimizing the mean and minimizing the CVaR. We quantify this as a *risk preference*, viz. a stipulation of the relative importances of the mean and the CVaR of the robot system. This yields a scalarized BiAP to seek a single optimal assignment that

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \big(\alpha \mu_{ij} x_{ij} + (1-\alpha)\text{CVaR}_{\lambda,ij} x_{ij}\big), \tag{5.6}$$

subject to (2.2)–(2.5) where $\alpha \in [0,1]$ is the risk preference. Thus, with this preference, one may apply a standard assignment algorithm to produce an assignment.

It is worth noting that λ is another parameter along with α that changes the scalarized objective value (5.6) because it changes the value of CVaR. Therefore, the determination of λ from $[0,1]$ should be taken into account for the scalarization. Conventionally, λ, which is the confidence interval with respect to CVaR, is chosen between 95% and 99%. In this work, we follow one of the conventions for determining λ, but varying its value is still a meaningful future direction to consider. We discuss an extension including the effect of λ in Sec. 5.4.4.

On the other hand, we note that some problems have an optimal assignment $\mathbf{X}^+$ such that

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\alpha \mu_{ij} x_{ij}^+ + (1-\alpha) \text{CVaR}_{\lambda,ij} x_{ij}^+ \right), \tag{5.7}$$

is minimized for all values of α in $[0, 1]$. We term these problems *risk preference indifferent* since the outcome is identical regardless of the risk preference value.

Theorem 5.16 Problems with costs of the form (5.1) subject to the six stated statistical requirements, with the additional fact the $\forall j\ (\forall i\ \mathsf{R}_{ij} = \mathsf{R}_i)$ and $\forall i\ (\forall j\ \mathsf{T}_{ij} = \mathsf{T}_j)$ (both R_i and T_j are random variables), and $\forall i, j\ \mathsf{E}_{ij} = k_{ij}$ (where each k_{ij} is constant) are risk preference indifferent.

Proof. If R_i has $\text{CVaR}_{\lambda,\mathsf{R}_i}$ and T_j has $\text{CVaR}_{\lambda,\mathsf{T}_j}$, then all assignments $\mathbf{X}$'s (optimal and otherwise) have a sum of $\sum_{i=1}^{n} \sum_{j=1}^{n} \text{CVaR}_{\lambda,ij} x_{ij}^+ = \sum_{i=1}^{n} \text{CVaR}_{\lambda,\mathsf{R}_i} x_{ij}^+ + \sum_{j=1}^{n} \text{CVaR}_{\lambda,\mathsf{T}_j} x_{ij}^+$. Hence, the optimal assignment only depends on the first term of (5.7), that is the one involving μ_{ij}. The minimizer of this assignment is the overall minimizer no matter the value of α. $\square$

Several practical problems fall into the category of risk indifferent. Fig. 5.2, for instance, is one such example, where the distributions are symmetric from the mean of robot or task locations toward the opposite directions. Since CVaR is proportional to the variability (e.g., variance) of a distribution, the sum of CVaRs of the two assignments A and B are the same. Thus, the value of α does not affect the determination of an optimal assignment.

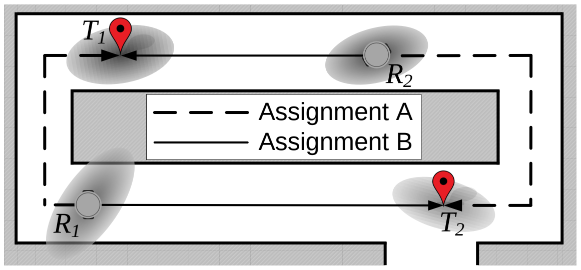


Figure 5.2: An example risk preference indifferent problem. Robots navigate to tasks via a road network. By virtue of the symmetric property of the normal distribution, $\mathsf{R}_{ij} = \mathsf{R}_i$ and $\mathsf{T}_{ij} = \mathsf{T}_j$ hold for all i and j. Assuming all E_{ij}'s are constant (or equal), two assignments A and B have the same variability, which makes them to have the same CVaR sum.

Nevertheless, certainly there remain many problems which are not risk indifferent. Since the value of α weights two sums with different meanings, it is a fairly *ad hoc* parameter. We therefore propose a sensitivity analysis that allows a user to determine how critical their choice of α is in producing the particular optimum. The analysis helps understanding the effect of the preference change and gives the user some help in determining its value.

5.4.1 Standard sensitivity analysis for the risk preference

Given a risk preference α, we compute a safe interval of α where any change within the interval does not change the current optimal assignment. Conceptually, this is the same as the sensitivity analysis introduced in Sec. 2.2.2. The difference is that the bounded region of costs is determined by the weighted sum of two objective values:

$$\mathbf{C} = \alpha \mathbf{M} + (1-\alpha)\mathbf{P}, \tag{5.8}$$

where $\mathbf{C}$ is a matrix representation of C_{ij} for $i, j \in \{1, \cdots, n\}$. Similarly, $\mathbf{M}$ and $\mathbf{P}$ are $n \times n$ matrices consisting of μ_{ij} and CVaR$_{\lambda,ij}$, respectively. The costs of a scalarized BiAP, $\mathbf{C}$ varies depending on α. Since α is a one-dimensional parameter in $[0, 1]$, (5.8) forms a line in the cost space.

As discussed before, $\theta(\mathbf{X}^*)$ is a polyhedral cone consisting of linear inequalities (2.6). Since we have a set of linear equations $\theta(\mathbf{X}^*)$ and a linear function $\mathbf{C}$, one can compute intersections between them. An intersection is a particular value of $\mathbf{C}$ that corresponds to a unique α because (5.8) is an injective function. Thus, we can compute an interval $\alpha \in [a, b]$ that is the minimum and maximum of those intersections. A straightforward way to achieve this is by linear programming where the objective function (5.8) is minimized or maximized over $\theta(\mathbf{X}^*)$. But because there is only one variable α, we do not need to consider the full cost space of $\mathbf{C}$. We may reduce the dimensionality to compute an interval of α, by minimizing and maximizing α over a set of linear inequalities that are functions of α, obtained by substituting c_{ij} by $\alpha\mu_{ij} + (1-\alpha)\text{CVaR}_{\lambda,ij}$.

Pseudocode for this procedure is shown in Alg. 7. Given $\mathbf{M}$ and $\mathbf{P}$, we compute $\mathbf{C}$ (line 1) and compute an optimal assignment by an assignment algorithm such as the Hungarian method (line 2). We use the sensitivity analysis described in Sec. 2.2.2 to compute $\theta(\mathbf{X}^*)$ (line 3). Then we rearrange $\theta(\mathbf{X}^*)$ as a function of α. Lines 4 and 5 compute the minimum and maximum intersection, so the objective values yield the exact interval of α where any α within the interval does not destroy the optimality of the current assignment.

Algorithm 7 RiskSA

Input: $n \times n$ matrices $\mathbf{M}$ and $\mathbf{P}$ that consist of μ_{ij} and $\text{CVaR}_{\lambda,ij}$, respectively. A risk preference α.

Output: An exact interval $[a, b]$ of α and the optimal assignment $\mathbf{X}^*$ in that interval.

```
1 $\mathbf{C} = \alpha\mathbf{M} + (1-\alpha)\mathbf{P}$
2 $\mathbf{X}^* = \text{HUNGARIAN}(\mathbf{C})$
3 $\theta(\mathbf{X}^*) = \text{SA}(\mathbf{X}^*, \mathbf{C})$ // compute $\theta(\mathbf{X}^*)$ by (2.7)
4 $a$ = LINPROG($\alpha$, $\theta(\mathbf{X}^*)$, 0, 1, max) // compute the max $\alpha \in [0, 1]$
5 $b$ = LINPROG($\alpha$, $\theta(\mathbf{X}^*)$, 0, 1, min) // compute the min $\alpha \in [0, 1]$
6 **return** $\mathbf{X}^*$, $a, b$
```

5.4.2 Heuristic sensitivity analysis for risk preference

The discussion in Sec. 2.2.2 anticipates the weakness of the preceding algorithm: the running time and space complexity are factorial in n because SA in line 3 of Alg. 7 enumerates all possible feasible solutions, a set whose size has factorial growth. In this section, we describe a heuristic method that produces estimates of the output interval much more quickly, whose pseudocode is described in Alg. 8.

The algorithm begins with a given value of α and computes the optimal assignment for α. The algorithm expands the interval for the optimal assignment downward (lines 5–14) and upward (lines 16–25). In each iteration, α increments or decrements by δ. It runs until the optimal assignment is altered with the changing α or a change of α reaches the boundary values (0 or 1). The algorithm underestimates the interval because it stops and does not expand the interval if a change by δ alters the optimal assignment. In each direction, the maximum difference between the approximated boundary and the exact boundary is δ. Thus, the approximated interval is smaller than the exact interval by at most 2δ. The time complexity of the algorithm is $O(n^3/\delta)$ since it calls the Hungarian method ($O(n^3)$) at most $\lfloor 1/\delta \rfloor$ times.

5.4.3 Extension: finding optimal assignments for $\alpha \in [0, 1]$

A practitioner may have some doubts when choosing a value of α. In such cases, it is useful to see how the optimal assignment changes for the entire range of values. This can be computed via sensitivity analysis of optimal assignments with respect to the value of α. This analysis can be achieved via repeated calls to Alg. 7. The first iteration finds an interval of an arbitrary α for its optimal assignment. Then the next iteration finds another interval outside the initial interval (i.e., choosing an arbitrary value of α outside the interval already found). The method finishes once all intervals in $[0, 1]$ are found. Since Alg. 7 finds the exact interval of α for an optimal assignment, choosing any α outside the

Algorithm 8 HeuristicSA

Input: $n \times n$ matrices $\mathbf{M}$ and $\mathbf{P}$ that consist of μ_{ij} and $\text{CVaR}_{\lambda,ij}$, respectively. A risk preference α_0 and the approximation parameter δ.

Output: An approximation interval $[a', b']$ of α and the optimal assignment $\mathbf{X}^-$ in that interval.

```
$\alpha = \alpha_0$
$a^- = b^- = \alpha$
$\mathbf{C} = \alpha\mathbf{M} + (1-\alpha)\mathbf{P}$
$\mathbf{X}^- = \text{HUNGARIAN}(\mathbf{C})$
while $\alpha - \delta \geq 0$ // expand $\alpha$ downward by $\delta$ in each loop
    $\alpha = \alpha - \delta$
    $\mathbf{C} = \alpha\mathbf{M} + (1-\alpha)\mathbf{P}$
    $\mathbf{X}^* = \text{HUNGARIAN}(\mathbf{C})$
    if $\mathbf{X}^* == \mathbf{X}^-$ // if $\mathbf{X}^*$ does not change with the new $\alpha$
        $a' = \alpha$ // expand $a'$
    else
        break // $\mathbf{X}^*$ changed, stop downward expansion
    end if
end while
$\alpha = \alpha_0$
while $\alpha + \delta \leq 1$ // expand $\alpha$ upward by $\delta$ in each loop
    $\alpha = \alpha + \delta$
    $\mathbf{C} = \alpha\mathbf{M} + (1-\alpha)\mathbf{P}$
    $\mathbf{X}^* = \text{HUNGARIAN}(\mathbf{C})$
    if $\mathbf{X}^* == \mathbf{X}^-$
        $b' = \alpha$ // expand $b'$
    else
        break // $\mathbf{X}^*$ changed, stop upward expansion
    end if
end while
return $\mathbf{X}^-$, $a', b'$
```

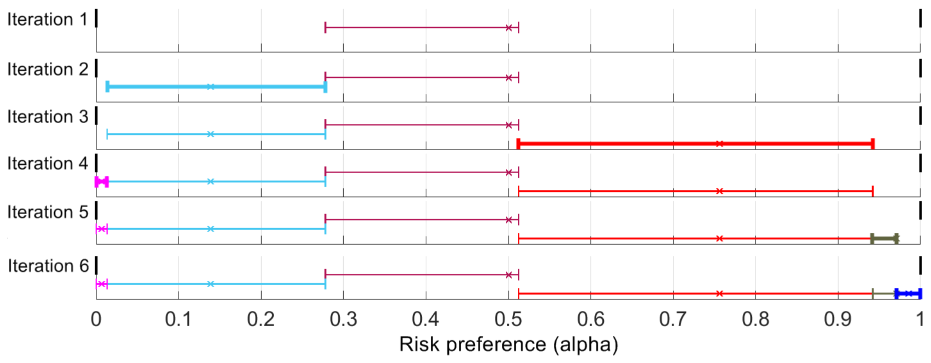


Figure 5.3: An example progression of Alg. 9 ($n = 4$). In each iteration, the algorithm finds a new interval (the thick horizontal bar) and its corresponding optimal assignment. The algorithm improves the map of α in each iteration and eventually finds the exact solution (Iteration 6).

range will give a new optimal assignment and its interval.

Although using Alg. 7 gives exact intervals of all optimal assignments for $\alpha \in [0, 1]$, it can be computationally intractable as the problem size increases (finding all Pareto frontiers is NP-complete). Thus, we repeat Alg. 8 until all optimal assignments for all discrete values of $\alpha \in [0, 1]$ (discretized by δ in Alg. 8) have been explored. This method may not find all optimal assignments for all continuous values of α, but give valuable information for determining α quickly.

Alg. 9 computes the sensitivity analysis of optimal assignments with respect to α using a modified version of Alg. 8 (a minor modification is limiting the working range of the algorithm to I instead of the full range $[0, 1]$) and Fig. 5.3 gives an example of use of the algorithm. The algorithm begins with the initial unexplored range of α which is $[0, 1]$ (line 2). It runs until no range remains unexplored or any of unexplored ranges is greater than δ. In each iteration, one unexplored range is selected and removed from the queue U (line 4). Then α is computed as the midpoint of the selected range (line 5). The optimal assignment $\mathbf{X}_i^*$ for that α and its approximation interval S_i is computed (lines 6, 7). In Fig. 5.3, the cross indicates α and the colored horizontal bar represents the interval. If the lower bound of the interval is larger than the lower bound of the previously unexplored range (line 8), a new unexplored range is inserted to the queue (line 9). Lines 11–13 show the same process for the upper bound. For example, $[a', b'] = [0.278, 0.512]$ in the first iteration in Fig. 5.3. Thus, $U = \{[0, 1]\}$ becomes $U = \{[0, 0.278), (0.512, 1]\}$. The time complexity of the algorithm is $O(n^3/\delta^2)$ since it calls Alg. 8 at most $\lfloor 1/\delta \rfloor$ times.

Algorithm 9 α-hSA

Input: $n \times n$ matrices $\mathbf{M}$ and $\mathbf{P}$ that consist of μ_{ij} and CVaR$_{\lambda,ij}$, respectively. The approximation parameter δ.
Output: A set of optimal assignments X and their α-intervals S.

```
1 $i = 0$
2 $U = \{[0, 1]\}$ //initialize the queue of unexplored $\alpha$
3 while ($U$ is not empty) || (any range in $U$ is greater than $\delta$)
4    $I = \text{DEQUEUE}(U)$ //dequeue one unexplored range
5    $\alpha = \text{MIN}(I) + \frac{\text{MAX}(I) - \text{MIN}(I)}{2}$
              //begin with the midpoint of an unexplored range
6    $[\mathbf{X}_i^*, a', b'] = \text{HEURISTICSA}'(\mathbf{M}, \mathbf{P}, \alpha, \delta, I)$ //run Alg. 8
7    $S_i = [a', b']$ //the apx. interval of $\mathbf{X}_i^*$
8    if $\text{MIN}(I) < a'$
9      $\text{ENQUEUE}(U, (\text{MIN}(I), a'))$ //exclude searched range
10   end if
11   if $\text{MAX}(I) > b'$
12     $\text{ENQUEUE}(U, (b', \text{MAX}(I)))$ //exclude searched range
13   end if
14   increment $i$
15 end while
16 return $X = \{\mathbf{X}_0^*, \cdots \mathbf{X}_\Pi^*\}$, $S = \{S_0, \cdots S_\Pi\}$
```

5.4.4 Remarks

In the BiAP, the optimal solution depends on α appreciably. Despite its importance, only empirical methods exist to select α. The preceding sections have provided sufficient conditions for problems in which the particular value of α is irrelevant. Secondly, they have presented algorithms that describe the importance of precise selection of α charting differing regimes of behavior because doing so reduces manual labor in exploring different risk policies.

On the other hand, λ is another parameter that decides the value of total scalarized cost as discussed in Sec. 5.3. In this chapter, λ is fixed to a predetermined value (i.e., one between 95% and 99%) since the parameter has a clear interpretation. If the determination of λ is not convinced, one could perform another sensitivity analysis with respect to the value of λ that is briefly described below.

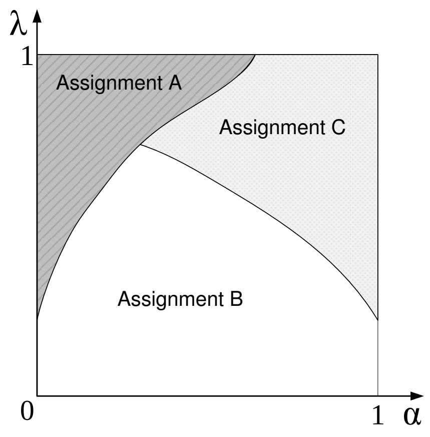
We know that (5.8) forms a line in the cost space while changing $\alpha \in [0, 1]$. Thus, given a fixed value of $\lambda \in [0, 1]$, changing $\alpha \in [0, 1]$ generates a line. For all values of λ, one ends of the lines (when $\alpha = 1$, which the CVaR term has no contribution to the cost sum) reach to the same point in the cost space. Changing both parameters produces a (hyper) surface in the cost space unless $\mathbf{P}$ in (5.8) is invariant (in this case, the cost still forms a line). The shape of the surface, which could be nonlinear, is determined by the cost distributions. The surface has a point that all constituting lines of the surface (i.e., each of the lines is formed when changing α for each value of λ) join at, such as a cone.

Given a cost surface, we can use the standard sensitivity analysis or RANDSA proposed in Ch. 4 to compute the regions of the optimal assignments on the surface. The regions from a sensitivity analysis correspond to the regions of optimal assignments on the α-λ plane as shown in Fig. 5.4, which is analogous to the one-dimensional mapping with respect to α in Fig. 5.3.[2]

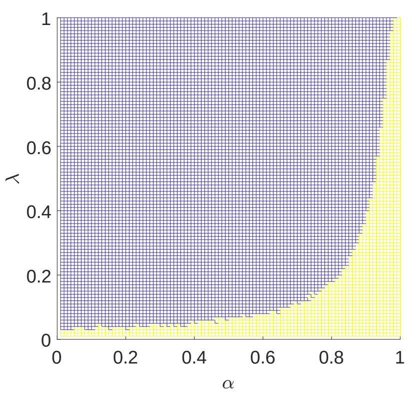
5.5 Experiments

We examine four distinct types of experiments. First, we generate random cost matrices that simulate the situations with large uncertainties. This experiment shows the

[2]Fig. 5.4b is not drawn by a sensitivity analysis but by changing two parameters manually.



(a) A hypothetical mapping.



(b) A mapping from a small-sized instance ($n = 5$).

Figure 5.4: A hypothetical and an example mapping of optimal assignments on the α-λ plane, which is analogous to the mapping with respect to α shown in Fig. 5.3.

validity of our formulation of using the mean and CVaR, which is a richer representation of costs than using scalar values only. Next, we measure the performance (running time and solution quality) of the algorithms with random instances. Next, we examine two robot scenarios where the mission is visiting task locations by robots (or self-driving vehicles), giving a view of the sensitivity of a risk preference in practice applications. First, we capture uncertainties in the state of robots from a robot simulator using extended Kalman filter (EKF) for localization. The result shows the usefulness of our algorithm for determining the value of α. Second, we collect traveling times in an urban area using Google Directions API [49]. The API provides a traveling time between two points for a specific time in a day. It is an instance of obtaining cost distributions from historical data or measurements. From the data, we compute the mean and CVaR of traveling times. The system is with Intel Core i3-2310M with 4G RAM and MATLAB R2015b.

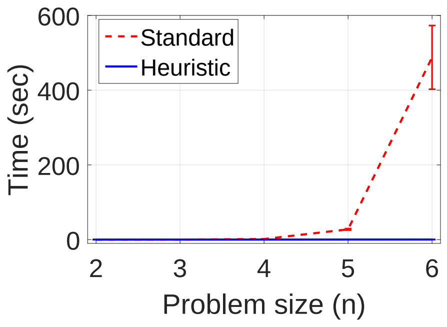
5.5.1 *Assignment computation considering uncertainties*

We run an experiment that shows the advantage of using the mean and CVaR than scalar values (e.g, mean) for costs. Our formulation is adequate for the situations with uncertainties, so we set cost distributions to have nontrivial variances. The probability distributions of costs are neither necessarily a particular distribution nor identical. We only need to know their density functions. Then the means and the CVaRs of those distributions can be computed. For the computations of CVaRs, the methods discussed in Sec. 5.2 can be used. Again, [75] provide closed-form equations of common CVaRs, so one may not need complex computations. We use normal distributions[3] for costs, where each cost has the mean and standard deviation drawn from uniform distributions $\mathcal{U}(0,10)$ and $\mathcal{U}(0,20)$, respectively (e.g., a normal cost distribution $\mathcal{N}(5,15^2)$). We randomly generate an $n \times n$ matrix of normal distributions and compute an optimal assignment based on both the mean and CVaR (with $\alpha = 0.05$, which is risk-averse). For comparison, we compute another optimal assignment only with the mean values. Then a cost matrix is sampled from the matrix of normal distributions. We compare the cost sum of the both assignments. We test 10,000 random matrices for $n = 50$ and set $\lambda = 0.95$. In average, our formulation reduces 7.511% of the cost sum (the standard deviation is 0.1578) compared to the case of using the mean only. For example, the cost sum of an assignment is 92.49 when the mean and CVaR are used whereas it is 100 when only the mean is used. The result shows that our formulation is able to save costs using the richer information regarding uncertainties.

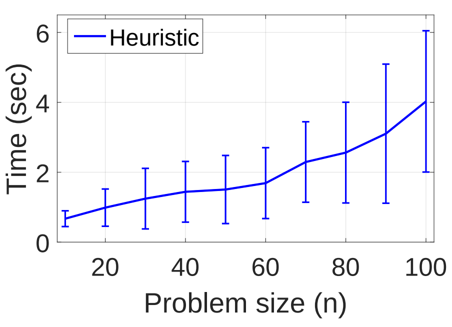
5.5.2 *Random instances for performance evaluation*

We randomly generate an $n \times n$ matrix of normal distributions where the mean and variance of each distribution are drawn from a uniform distribution $\mathcal{U}(0,1)$. The approximation parameter is $\delta = 0.001$ throughout all experiments. A smaller δ increases accuracy

[3]The CVaR of a normal distribution is $-\mu + \frac{\sigma}{\lambda} f(\lambda)$ for a pdf $f(\cdot)$.



(a) Running time up to $n = 6$.



(b) Running time for large instances.

Figure 5.5: Running time of random instances (statistics summarize 20 measurements). (a) The standard algorithm run until it finds the exact interval. (b) The running time of the heuristic method is shown for larger instances.

but increases the running time. The parameter δ can be adjusted considering the time allowed for the computation and the desired degree of accuracy of the solution.

Fig. 5.5, and Tables 5.1 and 5.2 show the results with those random matrices. The running time (averaged over 20 repetitions) of the standard method is prohibitive when $n > 6$. Up to $n = 6$, the standard and the heuristic method use the same problem instance for each repetition. For $n > 6$, we only run the heuristic method. The results in Fig. 5.5b show that the heuristic method is scalable for large instances.

Table 5.1: Running time (sec) of Alg. 7 and 8 (20 repetitions).

(a) Running time up to $n = 6$. The running time of the standard algorithm for $n > 6$ is prohibitive.

Alg.	n	2	3	4	5	6
Standard	Mean	0.09980	0.2030	1.724	27.59	487.7
	Std. dev.	0.09450	0.03960	0.05320	2.046	85.19
Heuristic	Mean	0.1465	0.2125	0.2479	0.3508	0.3961
	Std. dev.	0.04642	0.06633	0.08662	0.09421	0.1079

(b) Running time of the heuristic algorithm.

n	10	20	30	40	50	60	70	80	90	100
Mean	0.6706	0.9870	1.244	1.440	1.505	1.688	2.292	2.561	3.103	4.026
Std.	0.2255	0.5324	0.8664	0.8682	0.9744	1.014	1.151	1.441	1.990	2.021

We measure the running time and the solution quality of Alg. 9 for $n = 10, \cdots, 50$ (Table 5.2-a). This result is compared to a basic method that runs the Hungarian method iteratively from $\alpha = 0$ to 1. The number of iterations is determined to reach the same solution quality with Alg. 9 (i.e., α increments by 0.0007 in each iteration). The solution quality (Table 5.2-b) is measured by the searched ranges over the entire range of α. Notice that two methods are tested with the same set of random instances.

Alg. 9 shows faster running times than the iterative method while both methods produce high quality solutions. The algorithm runs offline before operating robots, so 15 seconds for 50 robots is reasonable.[4] The solution quality decreases as n grows because the methods with large instances calls their subroutines (i.e., Alg. 8 and the Hungarian method) frequently, where each call of a subroutine produces an error at most 2δ. The exactness of the solution is adjustable by δ depending on the given computation time.

Table 5.2: Results of Alg. 9 and an iterative method (20 repetitions).

(a) Running time (sec).

	n	10	20	30	40	50
Alg. 9	Mean	0.8717	2.532	5.061	9.073	15.51
	Std. dev	0.1736	0.5523	0.8147	0.9947	1.474
Iterative method	Mean	1.221	3.523	6.946	12.23	20.40
	Std. dev	0.2579	0.7551	1.090	1.096	1.981

(b) Solution quality (%).

	n	10	20	30	40	50
Alg. 9	Mean	99.37	98.81	98.35	97.71	96.74
	Std. dev	0.1000	0.2300	0.2700	0.4900	0.4400
Iterative method	Mean	99.73	99.29	98.89	98.47	97.77
	Std. dev	0.0900	0.1800	0.2200	0.3000	0.2700

[4]The algorithm computes of all optimal assignments for the entire range of α. Then a system operator (or robots) determines a particular value of α before execution. Or the value of α may change at run-time by looking at the result in hand.

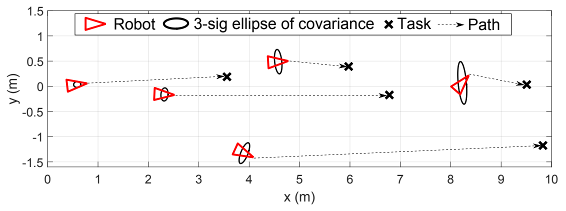
5.5.3 Cost uncertainties from state estimation

We use a robot simulator [26] employing a state estimator (EKF) to capture the uncertainties in robot poses so that costs can be represented by distributions. We place five robots and five tasks as shown in Fig. 5.6a where the mission is visiting the tasks by the robots without duplicated assignments. Cost is traveling time where the objective is to minimize the sum of traveling times. We assume that the robots move at 1 m/s (no acceleration/deceleration). Also, we assume that the locations of the tasks are certain, and the robots plan paths not to collide with other robots. For a robot-task pair, the path length between the mean robot position and the task location, which is the mean of the path length distribution, can be computed. The variance of the distribution is determined by the outgoing direction from the current robot pose (i.e., $\sigma^2 = \mathbf{v}^T\mathbf{\Sigma}\mathbf{v}$ where $\mathbf{v}$ is the vector representing the outgoing direction of a path and $\mathbf{\Sigma}$ is the covariance matrix). From this distance distribution, we calculate the cost (i.e., traveling time) distribution by dividing the mean and the standard deviation by the velocity. Repeating this computation for all n^2 costs constructs a matrix of cost distributions.

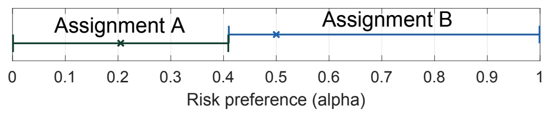
Fig. 5.6b shows the result from Alg. 9 where the input is the matrix of cost distributions and $\delta = 0.001$. Two assignments and their corresponding intervals are found. Assignment A is relatively risk-averse than Assignment B. It is shown that the determination of the value of α in the continuous space from zero to one is converted to choosing one assignment from the discrete set of two assignments, which is much simpler than the original problem.

5.5.4 Cost uncertainties from historical data or measurements

We examine a practical transportation problem of self-driving vehicles in a metropolitan area. We position five vehicles and five tasks at random locations on the map shown in Fig. 5.7. The mission is visiting the task locations by the vehicles where the traveling



(a) A snapshot from a robot simulator [26] with an example assignment of five robots and five tasks. The robots move to the tasks along the paths.



(b) The result from Alg. 9. Two assignments and their corresponding intervals are found.

Figure 5.6: An experiment with cost distributions from a state estimator. (a) Cost (traveling time) distributions are computed based on the path lengths and the covariance matrices of the robot poses. (b) Using Alg. 9, the problem of choosing α in the continuous space $[0, 1]$ is converted to choosing one assignment from the discrete set of assignments.

times vary depending on traffic conditions including physical interactions with other vehicles. We collect traveling times using Google Directions API [49]. For each pair of a vehicle and a task, 1,439 data points are collected, which are the measurements at every one minute during a day from 0:00 to 23:59. (The number of data points would seem small depending on the point of view, but a traveling time between two locations does not change radically in one minute.) Note that the path of a pair is the same for all data points, which is marked as an arrow in Fig. 5.7. These data reflect the traffic condition at the specific time in a day. Thus, the data points are naturally distributed.

Instead of fitting probability density functions to the data, we choose to use a nonparametric method to obtain the mean and CVaR for each vehicle-task pair. We calculate the mean from a set of data point for a pair. For the CVaR, we calculate the mean of the highest 5% of the set (i.e., $\lambda = 0.95$). This nonparametric method does not have the problem of imposing an incorrect assumption about the distribution.

We run Alg. 9 with the matrix of means and CVaRs of the traveling times. The algorithm finds a unique optimal assignment for any $\alpha \in [0, 1]$, which means that this problem belongs to the class of risk preference independent. The reason is that the CVaRs are proportional to the length of the paths (the means of the paths), so the optimal assignment

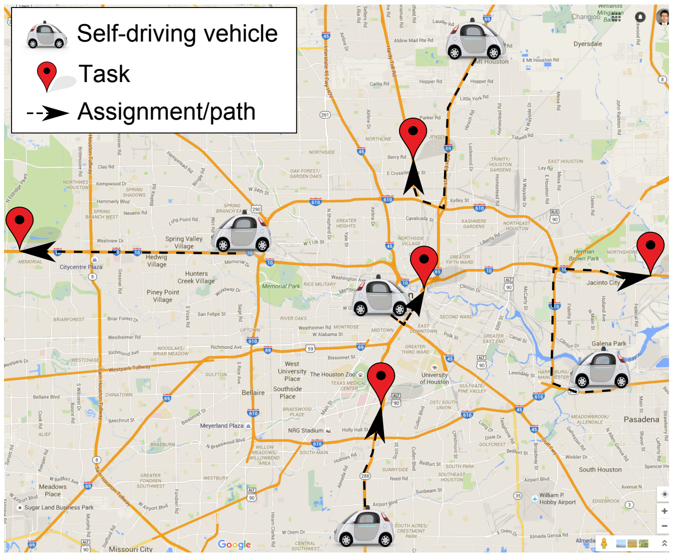


Figure 5.7: A transportation problem in a metropolitan area. The mission is visiting the tasks by the vehicles with the minimum sum of traveling times. The traveling times are distributed owing to the varying traffic conditions, so the scalar cost representation is not adequately rich.

does not change while α changes. Alg. 9 is able to decide whether a problem instance belongs to the risk preference independent class.

5.6 Summary

In this chapter, we consider multi-robot task allocation under uncertain costs. We use a cost representation incorporating uncertainty and interdependency, via distributional models. We provide conditions to show that the interdependencies among costs do not exist between elements in an assignment. The representation gives a new perspective on optimizing an allocation subject to a risk preference, where uncertainty takes a role in determining an optimal assignment. In addition, we show a problem class where the position taken on risk has no effect on the optimal assignment. For the problems where the risk preference is important, we provide algorithms for analyzing the sensitivity of an optimal assignment with respect to the risk preference. This enables a better understanding of how to determine the risk preference and its consequence. We would like to investigate further what conditions (i.e., properties of problem domains) make a task allocation problem risk indifferent.

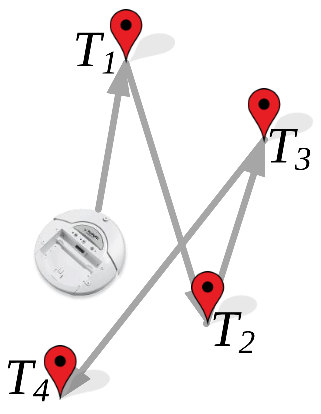
6. TASK BUNDLING FOR SEQUENTIAL STOCHASTIC TASKS IN MULTI-ROBOT TASK ALLOCATION

In the canonical formulation of the ST-SR MRTA problem, the sets of robots and tasks are fixed, and a decision-maker (e.g., a central computation unit) has full access to all information about the tasks. Practically, it may be impossible to know the complete set of tasks beforehand, as tasks might be revealed sequentially through observations or upon arrivals. Such *online tasks* are seen in many applications (e.g., dial-a-ride, material handling for online orders, demining). Compared to the case where the task set is known *a priori*, relatively little MRTA work examines online instances in a way that involves reasoning about the arrival of yet to be revealed tasks [53].

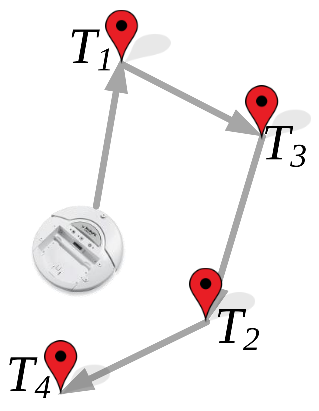
In this chapter, we study online tasks that are revealed sequentially and consider an infinite time horizon, with a focus on tasks with positive synergies. Serving tasks on an infinite time horizon needs a different performance metric other than the sum-of-cost which is conventionally used in MRTA. We consider two objectives which are the average system cost (e.g., fuel) per task and the average timespan of a task.[1] If multiple synergistic tasks are bundled together, then the total system cost of executing them is smaller than executing the tasks each independently. An obvious way to deal with these sorts of tasks is aggregating a set and then compute an allocation with the complete information for those tasks. Waiting for tasks to arrive in order to form a sufficiently large aggregation requires time which increases the timespan per task. This trade-off raises the question, *how many tasks should the robots bundle* over and above the standard question of *how should the tasks be allocated among robots*.

This chapter explores the foundations of multi-robot task allocation for tasks which

[1] In the domains where tasks are navigation, the cost and the timespan are equivalent to the distance traveled and the arrival time per task, respectively.



(a) The robot performs the tasks as soon as they are revealed.



(b) The robot waits for four tasks and performs them as a bundle.

Figure 6.1: A simple example: navigation tasks are revealed sequentially from T_1 to T_4. The robot performs tasks and loiters until a new task is introduced. In (a), the robot begins performing tasks right after they arrive. In (b), the robot waits until four tasks have been revealed, then finds a cheaper tour than (a).

arrive online, sequentially, and are synergistic in nature. We begin with a qualitative study with the basic setting where a task is revealed deterministically (e.g, within some fixed interval), where cost of the task being serviced is a function of its location, and the location of the task is independently and identically drawn from a known probability distribution. Later, we extend our scope of study to consider probabilistic task arrivals (a Poisson process) and non-i.i.d. task distributions. The set of tasks need not necessarily be bounded, but the set of robots is assumed fixed. Tasks arrive concurrently with execution of tasks, so tasks accumulate while robots are execute previous bundles. Fig. 6.1 depicts one iteration of a routing example, showing the marked effect of bundles with different numbers of tasks. We explore the basic case where robots work independently without coordination and tasks arrive deterministically with an i.i.d. spatial distribution. Next, using information that describes stochastic tasks, we model the objective values analytically as functions of bundle size. These models yield an optimal bundle size for each objective. However, using the predicted (constant) bundle size becomes suboptimal once the modeling the assumptions are violated (e.g., robots are coordinated so they have a different model of the system cost, or the tasks do not have a regular arrival interval, the task distribution is not i.i.d.) To address this, we propose simple policies that work optimally in the base case and that also efficiently adapt to improve their performance for more complex settings.

This chapter contributes a formulation of the problem of optimal task bundling for MRTA for sequentially revealed, synergistic tasks (Sec. 6.2.1). Within that formulation, we identify two objectives that describe two aspects performance, each depending on bundle size in a manner opposite the other. After analyzing the most basic scenario (Sec. 6.3.1), we introduce models that describe the objectives as a function of bundle size. Using these models, our study of iterated bundle executions leads us to propose simple and efficient bundling policies suitable for variations of the problem which generalize the basic instance. Evaluation of our policies (and comparison with a baseline sans-bundling) is carried out quantitatively with extensive experiments (Sec. 6.5).

6.1 Related Work

Most previous work in MRTA with online tasks focus on the question of *how to allocate tasks*. Early work on auction mechanisms [33] and greedy allocation [42] studied the allocation of online tasks where the total number of tasks is known. Some recent work [3, 78] considers online tasks with unpredictable arrivals, but the option of bundling is not discussed. Online bipartite matching algorithms (e.g., [103]), which solve the underlying mathematical problem of the online MRTA, also do not consider bundling. Instead, they study how to match online vertices based on their stochastic information.

While bundling tasks has not received much attention, we are certainly not the first to propose the idea. Koenig et al. propose Sequential Single-Item (SSI) auctions with bundles in [62]. In the bidding phase, robots submit bids for bundles (i.e., subsets) of tasks from a known set of all tasks. The bidding phase and the winner determination phase iterate sequentially until all tasks are assigned. Compared to the standard parallel auctions, this method reduces the team's cost by exploiting synergies among tasks. The approach also reduces the time spent bidding compared to the standard combinatorial auctions since not all permutations of assignments are considered. Zheng et al. [111] propose SSI with roll-outs where single tasks are auctioned in each iteration, but the cost of each task is evaluated together with the previously allocated tasks to each robot in order to exploit synergies. Heap and Pagnucco [54] extended SSI to bundles with Sequential Single-Cluster (SSC) auctions where the robots bid on clusters of tasks, formed through a k-means clustering algorithm.

Prior work with synergistic tasks uses two objectives MiniSum and MiniMax that are analogous to our *system cost per task* and the *timespan of a task*, respectively. These works have the same objective, namely, seeking the maximal synergy and the minimal timespan. However, the prior work considers a fixed, finite set of tasks and, as will become clear in the next section, considering an unbounded (and unremitting) sequence of tasks constrains the set of solutions because one must ensure that the task buffer does not overrun. In general, the prior work does not identify or have to address the consequences of bundling which worsens time-related objectives. But when tasks arrive in an online manner, there is a true trade-off that must be made; this aspect is absent from previous investigations. The present work also treats synergistic tasks in rather more sophisticated manner, providing an explicit model that quantifies the improvement in objective value.

Bullo et al. [20] survey the recent work on the dynamic vehicle routing (DVR) problem where visit locations are generated according to a spatio-temporal Poisson process. Various routing policies are described for different team properties (e.g., centralized or decentralized, communication capability) and constraints (e.g., deadlines, priorities, vehicle kinematics). Those policies are analyzed by spatial queueing theory, which provides the lower bound for the optimal system and the stability condition. This study can strengthen our work by providing theoretical analyses on the optimality of our task bundling policies in routing applications. However, the objective of our study is to identify principles for optimizing and coordinating robot systems that are domain independent. Thus, we study a numerical optimization of the bundle size, which is applicable whenever the tasks arrive online and ceaselessly.

6.2 Problem Description

This section formulates the problem mathematically, expresses constraints on the problem, and describes the objectives to be minimized. The multi-robot routing scenario is used as an example.

6.2.1 Problem formulation

Given a set $\mathbf{R} = \{R_1, \cdots, R_n\}$ of n robots, every $\alpha > 0$ time interval, a task T_j arrives and is enqueued along with other waiting tasks in a structure $\mathbf{T}$. The total number of tasks is unknown and it could be an unbounded sequence. Here α could be deterministic or a random variable—in the latter case, we use α to denote the mean of a distribution. We assume that robots share all available task information (i.e., $\mathbf{T}$) for example, through some communication network.

We model tasks by thinking about the costs associated with their performance—in order to make this concrete we will assume that the application entails mobile robots and the cost is a function of the locations of the robot and task. We assume that locations of tasks are drawn from a probability distribution, and this has the appropriate relationship on costs. Let $c(S)$ be the cost of performing the set of tasks S. We consider tasks with the property: for S_1 and S_2 where $S_1 \neq S_2$, $c(S_1) + c(S_2) > c(S_1 \cup S_2)$. In other words, performing multiple tasks together has the potential to lump some common work together and the cost of performing a bundle of tasks is sub-linear in bundle size, i.e., smaller than the sum of the costs when executed independently.

Robot R_i forms its own bundle $\mathbf{X}_i$ by extracting tasks from $\mathbf{T}$, where $|\mathbf{X}_i|$ would change. Once T_j is assigned to $\mathbf{X}_i$, it is no longer available to other robots unless R_i releases the task. Tasks continue to arrive while robots perform the work assigned to them, that is, that which is within their respective bundles. The robots iterate bundling and executing tasks in turn. Depending on the number of tasks in R_i's bundle, R_i may be idle while waiting to fill $\mathbf{X}_i$, which we denote $R_i \in \mathbf{R}_{\text{idle}}$. Otherwise, $R_i \in \mathbf{R}_{\text{active}}$. Note that $\mathbf{R} = \mathbf{R}_{\text{idle}} \cup \mathbf{R}_{\text{active}}$ and $\mathbf{R}_{\text{idle}} \cap \mathbf{R}_{\text{active}} = \emptyset$.

Strategies for assigning tasks to robots make use of flexibility in (i) making the choice of whom to assign to a certain task, and (ii) when to assign the task. In general, waiting increases the available opportunities to optimize performance but, waiting itself, induces delays. Since (ii) is a central consideration in the present work, it is important to delineate the requirements of the strategies for assigning tasks. We do this by noting two necessities for the performance of online tasks:

- *Unconditional Task Acceptance:* Any task that arrives, must be enqueued to $\mathbf{T}$.

- *Non-starvation:* No task may be abandoned to remain in $\mathbf{T}$ indefinitely.

Subject to fulfilling those two requirements, we consider two objectives to minimize. Since there is no fixed set of tasks in an infinite (or a very long) length horizon, the conventional sum-of-cost measure is no longer ideal. More meaningful are the average values of the following:

1. An important metric is the cost incurred by a robot to perform a task. Let $c_{ij} \in \mathbb{R}^{\geq 0}$ represent the cost of R_i performing T_j, then the objective is the average cost spent by a robot to finish a task, $\bar{c}$. The average is taken across all robots and task pairs; we call this the *system cost*.

2. The *timespan*, or *end-to-end time*, of a task $\tau_j \in \mathbb{R}^{\geq 0}$ metric represents the elapsed

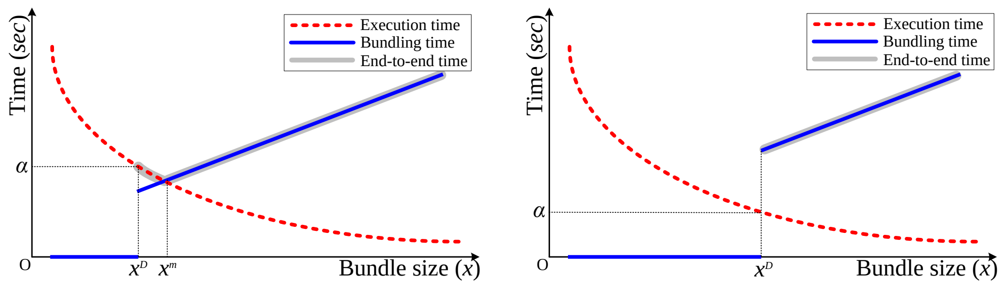
time from the moment task is inserted to $\mathbf{T}$ until its completion. The objective is the average timespan of a task, $\bar{\tau}$, taken across all tasks.

6.2.2 An example: the multi-robot routing problem

The multi-robot routing problem is a representative example of the setting we describe since it involves synergistic tasks that arrive online. It is also of natural interest since it includes features common to many application domains. Precisely, the tasks require that some robot visit a location. The locations are revealed sequentially and the goal is to visit all locations (as shown in Fig. 6.1) while minimizing the average time traveled $\bar{c}$, or the average task end-to-end time $\bar{\tau}$, or some combination of the objectives. A robot visits only one location at a time, and the task requires that one robot arrive. A robot bundling multiple tasks may then plan a Hamiltonian path. To solve the problem, the robots need a policy that determines how many locations they ought to bundle and how they should cooperate together to decide which robot should visit which locations. This is a running example throughout the chapter because it is sufficiently rich to explore the fundamental properties underlying strategic bundling.

6.3 An Analysis of Bundle Size

This section develops models for the objective values defined in the previous section. In the basic setting, robots work independently, and tasks are revealed with a fixed interval. For these simple models, it is possible to find analytic expressions for the optima. Next, we introduce some complexity into the model exploring, empirically, how coordination methods affect outcomes.



(a) The case where $x_g^* = x^m$.
(b) The case where $x_g^* = x^D$.

Figure 6.2: Illustrative functions that describe the average system cost (red) and timespan (gray) per task. The optimal bundle size for the system cost x_f^* is unbounded for both (a) and (b) since $f(\cdot)$ is strictly decreasing owing to the synergies among tasks. $s(\cdot)$ is infinite for $x < x^D$ and the same with $f(\cdot)$ otherwise. There exists a finite bundle size x that makes $g(\cdot)$ minimum. $g(\cdot)$ for $x < x^D$ is not shown since the value is infinite.

6.3.1 *The base case: independent robots*

6.3.1.1 *The general model*

In the base case, robots do not coordinate amongst themselves to exploit task locality *between* bundles, rather they bundle and execute tasks independently. Ordering of tasks *within* their bundles is optimized locally and depends on the path they construct. We assume stochastic tasks with locations independently and identically distributed from a Uniform distribution over a sub-region of the plane with area S. Also, a new task is revealed every α seconds. Steady-state models of both the system cost $\bar{c}$, which is the average system cost (execution time) per task, and the average timespan (end-to-end time) $\bar{\tau}$ of a task, are constructed next. It may be helpful to refer to Fig. 6.2 during the exposition.

A model for $\bar{c}$ is given by $f(x|S, v)$ where x is the bundle size and v is the task performance rate (e.g., velocity) of a robot (the red curve in Fig. 6.2). Task synergies imply that $f(\cdot)$ is decreasing and, hence, the bundle size that minimizes the system cost is infinite ($x_f^* = \infty$). The functional $h(x|\alpha, n, f)$ describes the average time that a task stays awaiting sufficient tasks to been enqueued to form a complete bundle. Since tasks are added into $\mathbf{T}$ and are distributed to n robots, $h(\cdot)$ increases when more robots are bundling, for a fixed α. But note also that $h(\cdot)$ is discontinuous since $h(x|\alpha, n, f) = 0$ if $x < x^D$ (the blue line in Fig. 6.2). Here the quantity x^D denotes the bundle size when $f(x|S, v) = \alpha$, that is, the point of equilibrium between the rate of task arrivals and (average) executions. Below x^D, $|\mathbf{T}|$ diverges so there is no steady-state (tasks keep being accumulated) and robot take tasks out from $\mathbf{T}$ without waiting, so the bundling time is zero. For $x \geq x^D$, tasks do not accumulate in the queue $\mathbf{T}$, and a robot must wait for tasks to arrive in order to fill its bundle and so the bundling time is nonzero. Thus, $h(x|\alpha, n, f) = h'(x|\alpha, n)$ for $x \geq x^D$, where $h'(\cdot)$ represents the bundling time without considering the potential overflow of $\mathbf{T}$. There is another component $s(x|\alpha, f)$, the residing time of a task in $\mathbf{T}$, which is the time spent by a task before any robot has returned and begun to assemble its next bundle. We have $s(x|\alpha, f) = \infty$ for $x < x^D$. Otherwise, $s(x|\alpha, f) = f(\cdot)$ since tasks only stay in $\mathbf{T}$ while robots are execute their bundles.

A model of $\bar{\tau}$ is given by $g(x|S, v, \alpha, f)$:

$$g(x|S, v, \alpha, f) = \max(f(x|S, v), h(x|\alpha, n, f), s(x|\alpha, f)), \tag{6.1}$$

which are the thick gray curves in Fig. 6.2. $g(\cdot)$ for $x < x^D$ is not shown since the value is infinite. The x value that makes $g(\cdot)$ minimum is the optimal bundle size x_g^*. Determining x_g^* consists of two possible cases, shown in Fig. 6.2a and Fig. 6.2b. In Fig. 6.2a, x^m is the equilibrium between the task bundling time and the execution time. At x^m, the robot finishes executing a bundle when the next bundle has just filled. Thus, $x_g^* = x^m$ because $g(x^m|\cdot)$ takes the minimum at this point. It is important to note that there is no waiting time between iterations. In Fig. 6.2b, x^m does not exist because the tasks in $\mathbf{T}$ overflow. At x^D, the execution time dominates the zero bundling time, and $f(x^D|\cdot)$ has the minimum, so $x_g^* = x^D$. Practically, x_g^* is computed by $\max(x^{m'}, x^D)$ where $x^{m'}$ is the intersection between $f(\cdot)$ and $h'(x|\alpha, n)$.

6.3.1.2 The multi-robot routing example

Since the pioneering work of Beardwood et al. [12], there has been extensive research on computing the optimal length of the tour in random instances of the traveling salesman problem (TSP). The early models in [12, 100] are simple but limited to the asymptotic behavior. Lee and Choi [67] proposed a more accurate model given a finite number of cities. The multi-robot routing problem aims to optimize the tour of each robot, which involves optimizing the Hamiltonian path (HP)—a special case of the TSP where the return tour to the start location is unnecessary. We modify the model in [67] for the HP. The system cost (time traveled) per task is

$$f(x|S, v) = \left(\frac{(0.7211\sqrt{x+1} + 0.604)\sqrt{S} - E_d}{v(x+1)} \right) \cdot (\beta \log x + 1) \tag{6.2}$$

where S is the area of a rectangular field and v is the velocity of the robot. The scalar E_d is the expected distance between two points drawn from a uniform distribution, representing the last visited location and the initial location (notice that the initial location is also random because it is the last visited location from the previous batch). The expression $0.7211\sqrt{x+1} + 0.604$ from the model in [67] yields the optimal length of a TSP tour with x locations,[2] and E_d is subtracted because a HP does not include the return trip to the initial location. The entire expression is divided by the number of locations (per task) and scaled by $\frac{\sqrt{S}}{v}$ (length/velocity). However, E_d is not scaled by $\sqrt{S}$ since it already includes the size of the area as a variable. A scaling factor, β, reflects how close the algorithm used is to optimal, where $\beta = 0$ for an optimal algorithm with larger values for practical suboptimal algorithms.

[2]The salesman starts at one of the locations, but the robot starts from the last location in the previous batch in our case. Thus, the total number of locations is $x + 1$.

A task waits in a bundle (size of x) for $x\alpha - j\alpha$ seconds where j is the time when the task is inserted. Then, the sum of the bundling time for all tasks is $\sum_{j=1}^{x} x\alpha - j\alpha = x^2\alpha - \frac{x(x+1)}{2}\alpha = \frac{\alpha}{2}x(x-1)$. And the function describing the bundling time per task is

$$h(x|\alpha, n, f) = \begin{cases} 0 & \text{if } x < x^D \\ h'(x|\alpha, n) = \frac{n\alpha}{2}(x-1) & \text{otherwise.} \end{cases} \tag{6.3}$$

Note that n is multiplied since tasks are distributed to n robots. Interestingly, $h'(\cdot)$ is a special case of the mean residual life of a customer in a renewal process presented in [61]. The residual life is the amount of time that the customer must wait until being served. The general form of (6.3) when task arrivals follow a Poisson process is

$$h'(x|\alpha, n, \sigma_\alpha^2) = \frac{n\alpha}{2}\left(1 + \frac{\sigma_\alpha^2}{\alpha^2}\right)(x-1) \tag{6.4}$$

where σ_α^2 is the variance of the arrival interval. If $\sigma_\alpha^2 = 0$, then (6.4) reduces to (6.3).

Fig. 6.3 shows (6.2) (red) as a function of bundle size (x) along with values from experiments (green) with values $\alpha = 10$, $v = 1\,m/s$, and $S = 150\,m \times 150\,m$. The blue line represents (6.3). We implemented a simple heuristic Hamiltonian path algorithm (Alg. 11 in Appendix D) to explore this model, and $\beta = 0.0542$ was empirically determined for our algorithm.

6.3.2 Coordinated robots

Next, we turn to coordinated robots. There are two major considerations in thinking about the objective values for teams of closely coupled robots. The first consideration is what task allocation method will be used to distribute tasks from $\mathbf{T}$ to the robots (e.g.,

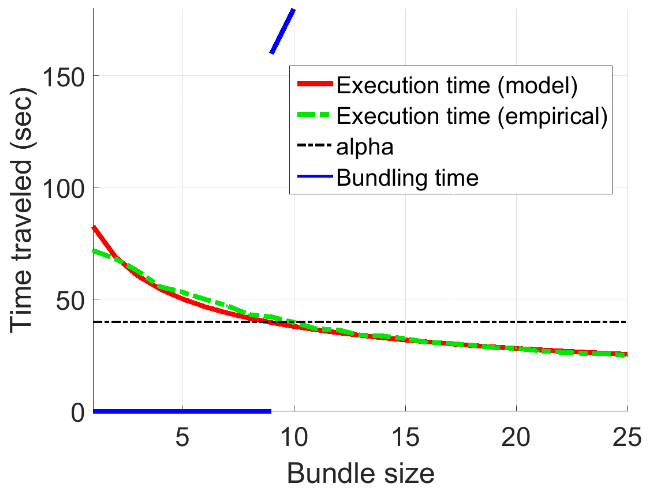


Figure 6.3: The models (6.2) (red dotted) and (6.3) (blue solid). The horizontal line represents α. The greed curve shows the experimental result from a heuristic HP algorithm (Alg. 11 in Appendix D).

assignment algorithms, integer programming methods, auction-based algorithms, or etc.)[3]. Our implementation uses integer linear programming (ILP) to optimally distribute tasks based on the distances between robots and tasks. The second consideration is the degree of synchronization in the team for the task distribution. Specifically, we need to decide how many robots are included when distributing tasks. If the robots that have completed their tasks wait until other robots become free, the distribution of tasks among them can make maximal use of their spatial dispersion. If the robots do not wait, the tasks they are assigned will suit them individually, being slightly myopic. Waiting for robots to finish their tasks allows more robots to participate in the assignment, imposing greater synchronization on the robots. But, in a way analogous to the advantages of large task bundles over small ones, it gives more opportunity for the optimizer to find savings.

Modeling this coordinated case is possible when one has domain knowledge of the sort used to build the model in Sec. 6.3.1.2. However, the necessary domain knowledge is not always available so, as an alternative, one may fit a function to empirical data. We are not

[3]In some domains, combinatorial approaches have been studied to solve the task distribution and optimization together (e.g., the multiple TSP [13]).

aware of any model describing the system cost of the coordinated robot team in the multi-robot routing problem. Fitting a function may require extensive experiments be performed, which may be tedious or expensive. Depending on the coordination method used, one can still draw inferences on how the system cost changes by examining the model of the base case, and making adjustments for coordinated case from a empirical data.

In Fig. 6.4, we show measured values of $f(\cdot)$ when robots are coordinated through an ILP (averaged over 10 repetitions). The x-axis and y-axis represent the bundle size (x) and the degree of synchronization (n_{ILP}). Given 5 robots, n_{ILP} ranges from 1 (completely asynchronous) to 5 (completely synchronized). Fig. 6.4b shows the system cost across different bundle sizes. The uppermost line (lime green) shows the cost when $n_{\text{ILP}} = 1$. The lower-most line (brown) describes the case where $n_{\text{ILP}} = 5$. The result shows that a synchronized team outperforms asynchronous robots[4]. There are two reasons why synchronous robots perform better even though additional idle time is incurred in waiting for other robots. First, as already alluded to, including more robots in the ILP makes the resulting assignment globally optimal with the current tasks in $\mathbf{T}$. Secondly, while the robots wait for other robots, they are simultaneously getting more options for tasks because new ones keep arriving. Since the ILP includes all tasks in $\mathbf{T}$ and $\mathbf{R}_{\text{idle}}$, the chance of lowering the cost per task increases as $\mathbf{T}$ grows.

On the other hand, there appears to be an anomaly in Fig. 6.4 as, when x is small, the cost does not decrease monotonically with increasing x. It indicates that bundling two tasks is worst than not bundling them. The transition between two lines in Fig. 6.4c comes from the synchronization and the optimal task distribution. The oddity is not observed in the independent and synchronized robot team. It is the optimal task distribution which explains the oddity. When the bundle size is small, tasks overflow $\mathbf{T}$. Their locations are

[4]The result from the independent robots case is not shown, but a synchronized team performs better than it too.

uniformly distributed in space, so that as $\mathbf{T}$ overflows, robots are able to find one or two tasks with very small costs. As the bundle size increases, tasks are removed more quickly and the density of tasks decreases, and the distance per task inevitably returns to normal. In the independent team, the tasks are randomly distributed to robots so the dense tasks make for no change.

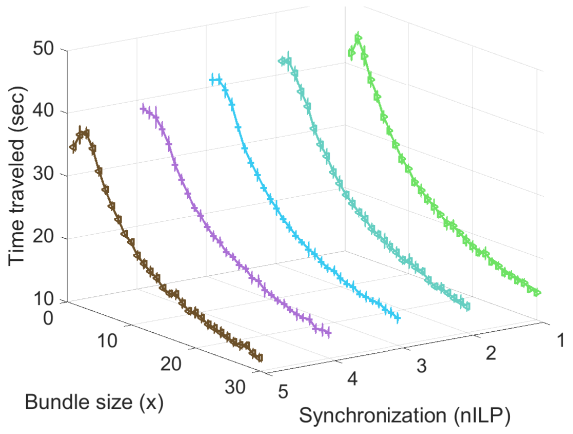
By bringing these insights together one comes to the conclusion that, for the setting being examined, the base case is an upper bound of the system cost, across all combinations of coordination methods and degree synchronization. Fig. 6.4c shows the base case (upper black line) and the coordinated robots case where $n_{\text{ILP}} = 5$ (lower brown line). Thus, the base case provides a model that overestimates cost in other cases. If we compute x_g^* using the basic model, x_g^* is larger or equal to the actual optimal bundle size (e.g., the red curve in Fig. 6.2 moves below, so x^m or x^D decrease). Without deeper domain knowledge, one cannot know the exact x_g^* but the range is determined from the basic model.

6.3.3 Elements of task stochasticity

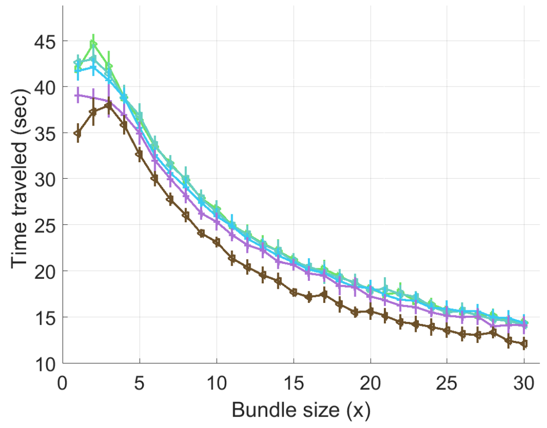
Tasks locations and arrivals include uncertain elements that induce a gap between the model and the performance observed in the system. Next, we address the issues arising from stochasticity in tasks; it motivates our introduction of bundling policies that adapt to circumstances, and can be useful when the ideal models fails to capture some aspects of the system.

6.3.3.1 Task locations

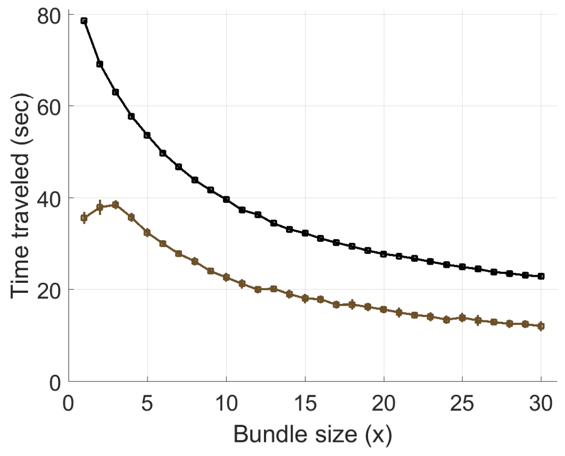
Owing to the stochasticity of task locations, the system's cost described thus far should be those of as describing the mean of a random variable. The particular realizations will differ from this average value. In practice, if a robot completes its bundle of tasks faster than the average execution time, the next bundle will not be completely filled yet. Thus the actual x_g^* would differ from the value in the basic model.



(a) The system cost along x and n_{ILP}.



(b) The system cost of different n_{ILP} along x.
(This view simply projects n_{ILP} out of Fig. (a).)



(c) The comparison between the base case and the case of coordinated robots with $n_{\text{ILP}} = 5$.

Figure 6.4: Empirical results of a team of five robots. In (b), it is shown that synchronization improves the performance (the uppermost is the case where $n_{\text{ILP}} = 1$ and the lower-most has $n_{\text{ILP}} = 5$). (c) shows that the basic model from Sec. 6.3.1 is the upper bound of all combinations of coordination methods, synchronization.

6.3.3.2 Task arrival process

Though some applications certain have tasks are revealed at fixed intervals, a probabilistic arrival process is more common. One generalization is to consider a Poisson arrival process, modeling completely random arrivals of events. Probabilistic arrivals will alter the true x_g^* so that it differs from the one in the basic model because $h(\cdot)$ is no longer deterministic (e.g., the blue line in Fig. 6.2 has some variance).

6.4 Bundling Policies

Based on the previous discussion, this section proposes some bundling policies, while the following section provides through evaluations. Each of the policies are simple algorithms that perform optimally in the basic case, but provide broader support, being flexible enough to behave agreeably across a range of more complex cases.

6.4.1 Model-based policies

6.4.1.1 Fixed-x policy

If the models $g(\cdot)$ and $h(\cdot)$ are available, finding an x that minimizes (6.1), the end-to-end time, gives x_g^*. Also, finding the x that minimizes $h(\cdot)$, the system cost, gives x_f^*. Let k be the bundle size that each robot takes (it may differ for different robots if their performance rates differ, such as having different velocities). Each robot keeps $k = x_g^*$ or $k = x_f^*$ depending on its objective. Since x_f^* goes to infinity with synergistic tasks, this is not a practical bundle size. Thus, we only consider $k = x_g^*$. When a robot finishes its current bundle and tries to execute the next bundle, there might be insufficient tasks in $\mathbf{T}$ to form that bundle. The robot may wait, idly, for new tasks. Let x_{P} denote the size of $\mathbf{T}$ which triggers execution of the current bundle. In the fixed-x policy, $x_{\mathrm{P}} = x_g^*$. If $|\mathbf{T}| \geq x_{\mathrm{P}}$, the robot takes k tasks and executes them immediately. Because this policy cannot handle uncertainties in the task profile (discussed in Sec. 6.3.3) it is possible that end-to-end time could diverge if $\mathbf{T}$ overflows.

6.4.1.2 Up-to-x policy

This policy is the same with the fixed-x policy except that the robot does not wait to fill its bundle. In every iteration, the robot takes k tasks from $\mathbf{T}$, where $k \leq x_g^*$, if $|\mathbf{T}| \geq x_{\mathrm{P}} = 1$. This eliminates the bundling time.

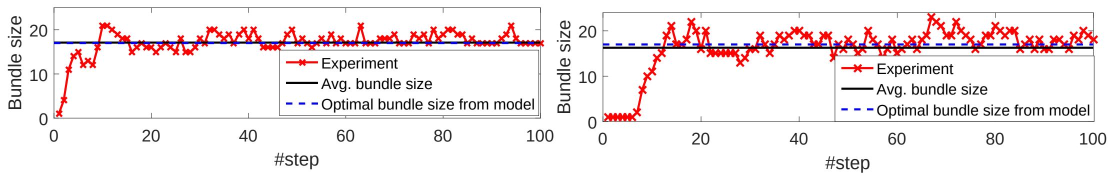
6.4.2 Remarks

As we discussed in Sec. 6.3.2, when the robots are coordinated and/or synchronized, then the x_g^* computed from the basic model differs from the actual optimal bundle size. Since x_g^* overestimates the true value, the consequence is that it may increase the end-to-end time to some degree, but it never causes $\mathbf{T}$ diverge. The sub-optimality in end-to-end time is somewhat allayed by having a better system cost (from the larger task bundles).

6.4.3 Model-free policies

6.4.3.1 Sweeping policy

The sweeping policy takes all tasks $k = |\mathbf{T}|$ if $|\mathbf{T}| \geq x_{\mathrm{P}} = 1$. Since the previous two polices cannot handle the case where the robots complete their current bundles earlier than the expectation, or tasks arrive faster than the mean interval. These two types of behavior may happen due to fluctuations away from the mean for some period of time. This policy saves bundling time when the number of tasks is insufficient. Also, the policy exploits the synergies maximally by taking all available tasks. Most importantly, the policy is useful even when the models are unavailable. The bundle size converges to a value reflecting the equilibrium in which the execution time and bundling times are equal (in average). Fig. 6.5a shows the changes of the bundle size versus time when task locations are uniformly distributed and tasks arrive regularly.



(a) The sweeping policy (b) The averaging policy

Figure 6.5: Plots showing bundle size vs. iterations. The policies converge to the optimal bundle predicted from the model.

6.4.3.2 Averaging policy

The sweeping policy's equilibrium is constant unless the stochastic properties of the task location and arrival process change. The sweeping policy does not make explicit use of a representation of the equilibrium. It is worthwhile to modify to track the equilibrium via history. The averaging policy begins with $x_{\mathrm{P}} = 1$ and repeats the following: if $|\mathbf{T}| \geq x_{\mathrm{P}}$, then the robot records the current $|\mathbf{T}|$ in the history window W (i.e., $W \leftarrow \textsc{Enqueue}(W, |\mathbf{T}|)$). It then takes $k = x_{\mathrm{P}}$ tasks and leaves the remaining tasks behind in $\mathbf{T}$. Next, a new x_{P} is computed by averaging the previous values saved in W (i.e., $x_{\mathrm{P}} = \textsc{Mean}(W)$). The smaller the window size, the more sensitive the policy to variability.

6.4.4 Algorithm

We develop a multi-robot routing algorithm with a centralized task distribution mechanism. Pseudocode appears in Alg. 10, using of the fixed bundle-size policy. In the initialization (lines 1–8), the bundle $\mathbf{X}_i$ of robot i is filled with the current location of the robot, and the optimal bundle size x_i^* is computed. After the initialization, the algorithm iterates lines 10–31 infinitely. Lines 10–17 run for those robots that execute their Hamiltonian paths. Line 11 executes the path and removes visited locations from the bundle. A robot transitioning into the idle state in lines 12–16. The algorithm allocates tasks to idle robots in each step (line 19). If there are available tasks and idle robots, the bundles are updated through ALLOCATE function.

ALLOCATE computes an optimal assignment of tasks in $\mathbf{T}$ to the idle robots in $\mathbf{R}_{\text{idle}}$, where $|\mathbf{T}| = m$ and $|\mathbf{R}_{\text{idle}}| = n'$. One or more tasks are assigned to each robot based on the cost c_{ij} where $c_{ij} = ||l(R_i) - l(T_j)||$. Let y_{ij} be a binary variable that equals to 0 or 1, where $y_{ij} = 1$ indicates that R_i performs T_j, and $y_{ij} = 0$ elsewhere. Then a mathematical description of the assignment problem is

$$\min \sum_{i=1}^{n'} \sum_{j=1}^{m} c_{ij} y_{ij} \tag{6.5}$$

subject to

$$\sum_{i=1}^{n'} y_{ij} = 1 \qquad \forall j, \tag{6.6}$$

$$y_{ij} \in \{0, 1\} \qquad \forall \{i, j\}. \tag{6.7}$$

(6.6) prohibits a task to be assigned to multiple robots.

After computing the assignment, $\forall j$, with $y_{ij} = 1$, the associated T_j is added to $\mathbf{X}_i$. Those assigned tasks are removed from $\mathbf{T}$ (line 20). Lines 22–31 are policy dependent. The lines run for all idle robots, which wait until the condition for the policy is satisfied. After that, robot i becomes active, computing a Hamiltonian path from the locations in its batch (lines 27–29).

6.5 Quantitative Study: Comparisons of the Policies

In this section, we describe experiments that examine the various polices on the multi-robot routing problem. The coordination method called 'IND' randomly distributes tasks from $\mathbf{T}$ to robots. As to the dimension of synchronization: asynchronous robots do not

Algorithm 10 M-HP

Input: the number of robots n, the velocity of robots v_i, the area of the field S, the task arrival interval α

Output: Continuous executions of $\mathbf{X}_i$

```
1 $\mathbf{T} = \emptyset$//the set of tasks
2 $\mathbf{R}_{\text{active}} = \emptyset$//the set of working robots
3 $\mathbf{R}_{\text{idle}} = \{R_1, \cdots, R_n\}$//the set of idle robots
4 $\mathbf{TP}_i = \emptyset$//a temporal set
5 for each $R_i \in \mathbf{R}_{\text{idle}}$
6   $\mathbf{X}_i = \{l(R_i)\}$//$l(\cdot)$: the location of the input
7   Compute $x_i^* = \max(x^m, x^D)$
8 end for
9 while uninterrupted
10  for each $R_i \in \mathbf{R}_{\text{active}}$
11     Execute $\mathbf{X}_i$ and remove visited locations
12     if $\mathbf{X}_i = \emptyset$//no more location to visit
13        $\mathbf{R}_{\text{active}} = \mathbf{R}_{\text{active}} \setminus R_i$
14        $\mathbf{R}_{\text{idle}} = \mathbf{R}_{\text{idle}} \cup R_i$
15        $\mathbf{X}_i = \{l(R_i)\}$//the start location of next $\mathbf{X}_i$
16     end if
17  end for
18  if $|\mathbf{R}_{\text{idle}}| \geq n_{\text{ILP}}$
19     $\mathbf{X}_{1,\cdots,|\mathbf{R}_{\text{idle}}|}$ = ALLOCATE($\mathbf{T}$, $\mathbf{R}_{\text{idle}}$)//solve (6.5)-(2.5)
20     $\mathbf{T} = \mathbf{T} \setminus (\mathbf{X}_1 \cup \cdots \cup \mathbf{X}_{|\mathbf{R}_{\text{idle}}|})$//remove allocated tasks
21  end if
22  for each $R_i \in \mathbf{R}_{\text{idle}}$
23     $\mathbf{TP}_i = \mathbf{TP}_i \cup \mathbf{X}_i$//merge the bundle with
                      //the remaining tasks in $\mathbf{TP}_i$
24     if $\mathbf{TP}_i \geq x_{\text{P}}$//$x_{\text{P}}$: policy-dependent parameter
25      $\mathbf{X}_i = \mathbf{TP}_i(1:k)$//$k$: policy-dependent bundle size
26      $\mathbf{TP}_i = \mathbf{TP}_i(k+1:\text{end})$
27      $\mathbf{X}_i$ = RANDHP($\mathbf{X}_i$)//compute an HP
28      $\mathbf{R}_{\text{idle}} = \mathbf{R}_{\text{idle}} \setminus R_i$
29      $\mathbf{R}_{\text{active}} = \mathbf{R}_{\text{active}} \cup R_i$
30     end if
31  end for
32 end while
```

wait for other robots (but may be included with other robots by chance). In contrast, synchronized teams works as a block. We also include numbers from a baseline where robots do not bundle but instantaneously execute tasks one by one.

6.5.1 Experimental settings

For a fixed number of robots ($n = 5$), we assume that all robots move at the same velocity v, and the bundle size is computed from the model is the same for all robots. As discussed above, using x_f^* is unrealistic since it would make robots wait for an unbounded number of tasks. Thus, we minimize the end-to-end time only, and scrutinize how the system cost changes. We set $\alpha = 5$ for the regular task arrival process. The parameter for the Poisson arrival process is $\rho = \frac{1}{\alpha}$, where the mean arrival interval is $\rho^{-1} = \alpha$. Those two arrival processes have the same mean task arrival interval. We measure the two objective values and run 10 repetitions. The results appear in Table 6.1 and Fig. 6.6.

6.5.2 Analysis

The results show that all bundling polices in all combinations represent significant improvements over the non-bundling baseline. The improvement in the time traveled is a consequence of larger bundles although we aim to optimize the end-to-end metric. We focus on the end-to-end time in the following analysis. Table 6.1a and Table 6.1b show the results of the fixed task arrival interval case and the Poisson arrival case, respectively.

In both cases, synchronizing the robots (Sync) with an optimal task distribution mechanism (AS) significantly increases performance. As discussed, this is because including all robots in AS results in a globally optimal assignment solution. If the robots bundle tasks, Sync gains more tasks while the robots wait for each other. This increases the chance of having lower-cost tasks in AS. Bundling improves the performance as well. Bundling reduces the execution time which is one of the components of the end-to-end measure (the residing time in $\mathbf{T}$, the bundling time, and the execution time). If robots finish tasks

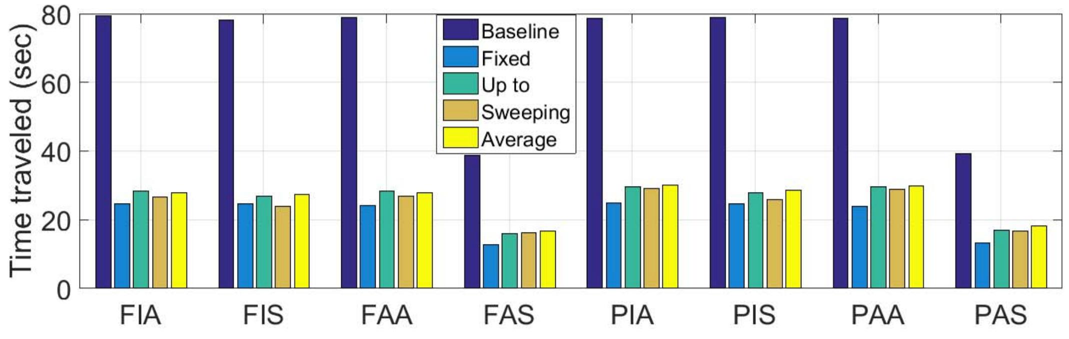
Table 6.1: Comparisons of policies. The values represent the mean and the standard deviation over 10 repetitions.

(a) Fixed task arrivals ($\alpha = 5$)

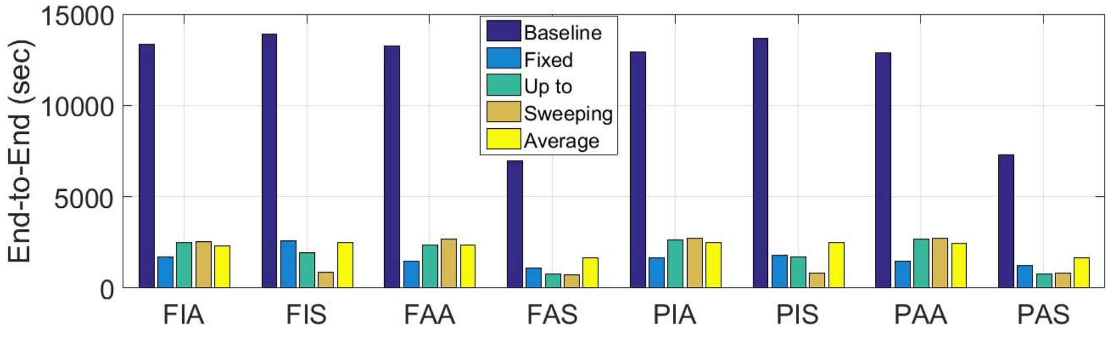
	Deg. of Sync	Time traveled		End-to-end	
		IND	AS	IND	AS
Baseline	Async	79.31 (1.073)	78.77 (1.135)	13300 (223.7)	13210 (212.0)
	Sync	78.09 (1.088)	38.53 (0.9862)	13890 (340.1)	6939 (479.8)
Fixed x	Async	24.60 (0.3407)	24.07 (0.1681)	1696 (107.6)	1434 (65.54)
	Sync	24.45 (0.3596)	12.70 (0.5858)	2557 (177.7)	1059 (166.2)
Up to x	Async	28.27 (0.5257)	28.25 (0.4188)	2476 (166.6)	2344 (163.1)
	Sync	26.84 (0.1947)	15.74 (0.4869)	1926 (165.0)	726.3 (126.0)
Sweeping	Async	26.58 (0.1870)	26.72 (0.2104)	2499 (160.5)	2668 (201.7)
	Sync	23.78 (0.6062)	16.05 (0.9959)	834.9 (51.30)	714.96 (192.1)
Averaging	Async	27.63 (0.4369)	27.79 (0.2985)	2266 (184.6)	2321 (116.9)
	Sync	27.18 (0.8579)	16.60 (0.8263)	2455 (151.3)	1650 (127.6)

(b) Poisson task arrivals ($\rho = 1/5$)

	Deg. of Sync	Time traveled		End-to-end	
		IND	AS	IND	AS
Baseline	Async	78.49 (1.111)	78.59 (0.7917)	12910 (220.7)	12880 (152.4)
	Sync	78.65 (0.7509)	39.24 (1.301)	13670 (304.0)	7264 (652.2)
Fixed x	Async	24.65 (0.2831)	23.83 (0.3252)	1613 (109.7)	1434 (70.31)
	Sync	24.49 (0.5503)	13.03 (0.5994)	1787 (322.0)	1216 (286.6)
Up to x	Async	29.48 (0.6724)	29.56 (0.4917)	2610 (195.0)	2639 (257.5)
	Sync	27.75 (0.6123)	16.91 (0.6896)	1688 (129.4)	760.3 (199.5)
Sweeping	Async	29.02 (0.6002)	28.76 (0.5024)	2702 (236.6)	2686 (231.7)
	Sync	25.75 (0.4472)	16.66 (0.9694)	801.4 (51.73)	812.8 (208.1)
Averaging	Async	29.97 (0.5120)	29.73 (0.5907)	2475 (190.7)	2402 (216.3)
	Sync	28.59 (1.404)	17.98 (0.8220)	2473 (158.5)	1615 (227.2)



(a) The time traveled (the system cost).



(b) The end-to-end time (the timespan of a task).

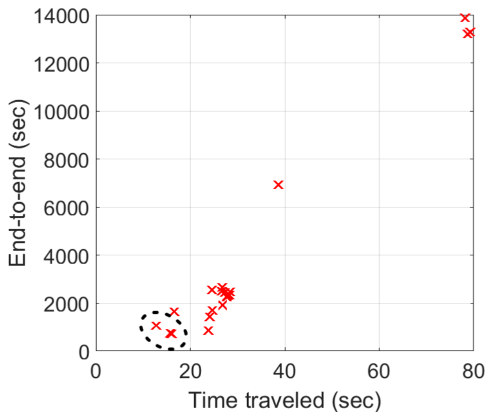
Figure 6.6: Comparisons of policies with all combinations of the arrival process (Fixed or Poisson), the coordination method (Independent or Assignment), and the degree of synchronization (Async or Sync). Three letters represent the combination.

faster, the residing time in $\mathbf{T}$ decreases, which is also a component of the end-to-end time. A larger bundle increases the bundling time only, and this increase is dominated by the decrease of other two components.

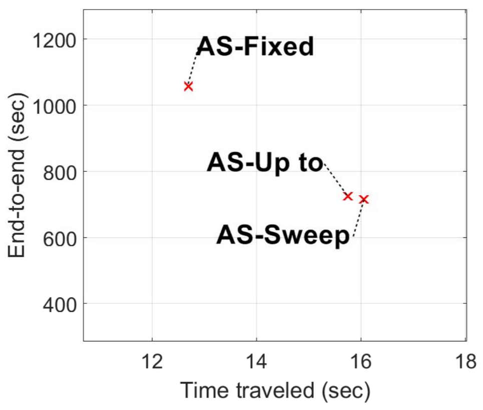
Bundling outperforms instantaneous executions. Also, with regard to coordination method: Sync outperforms Async, and AS outperforms IND. Using only Sync does not always guarantee an improvement. If we use Sync, robots sometimes wait too long to fill the bundle. This bundling time becomes overshadows other times in Sync and IND because the bundles of all robots should be filled, and IND does not take maximum advantage of synchronized robots (i.e., there is no optimization in task distribution). Using only AS also does not always guarantee a significant improvement. Using AS with Async has robots that are quiet close to working independently. It is unlikely that robots will finish their bundles at the same time, so most time they individually take lower-cost tasks. In a bounded region, visiting a bundle of random locations versus only those locations close to the robot may have negligible difference, especially when we optimize the tour.

The fixed-x policy yields the smallest system cost. The policy never takes fewer tasks than its overestimate of the optimal bundle size, and it exploits synergies among tasks. The up-to-x policy reduces the bundling time by never waiting to fill bundles, but this may increase the system cost. The sweeping policy is similar to the up-to-x but without limiting the number of tasks that the robots bundle. Therefore, the system cost is better than up-to-x, in general. The averaging policy does not show any remarkable performance; its advantage is in dealing with noisy patterns in the task profile.

In choosing a policy, one's purpose must be borne in mind. To help determine the appropriate policy, we show the results in the objective space in Fig. 6.7. Among all combinations (in Fig. 6.7a), only non-dominated combinations are shown in Fig. 6.7b. The Pareto frontier is consists only of combinations of AS and Sync, so coordinating and synchronizing robots is the most beneficial for both objectives. Beyond this combination,



(a) All solutions.



(b) The non-dominated solutions.

Figure 6.7: The objective space of the case where tasks arrive with α. The polices form a Pareto frontier as marked in (a). The three polices correspond to the fixed-x, up-to-x, and the sweeping policy with the synchronized robots using the ILP. The Poisson arrival case is omitted since it has the same result.

one must select a policy. For the system cost, we would use the fixed-x policy if models are available. Without any model, the sweeping policy is the best. For the end-to-end time, the up-to-x and the sweeping policy show similar performances. We would choose between them according to the availability of the models.

6.5.3 Non-i.i.d. task locations and task arrival interval

We also run experiments with the task locations that are *not* independently and identically distributed. With a probability of 0.5, a task is drawn from a Uniform distribution within the arena that robots work. With a probability of 0.5, a task is drawn from a Normal distribution that has the location of the last task as the mean. The task arrival process also has the interval between tasks that is non-i.i.d. With a probability of 0.5, the arrival process follows the Poisson process with ρ which is a sinusoidal function. With a probability of 0.5, the interval is drawn from a Uniform distribution where the upper bound is related to the previous value of ρ.

We only report results for Sync and AS. We tested the sweeping and the average policies since the models are not available in this non-i.i.d. case. The results (Table 6.2) show that the model-free policies outperforms the baseline method. Among the two model-free policies, one would choose the sweeping policy since this policy is shown to adapt itself well to changes in the task profile.

Table 6.2: The results from non-i.i.d. task locations and arrival intervals. Sync and AS are used.

	Time traveled	End-to-end
Baseline	37.64 (0.7788)	5856 (293.1)
Sweeping	14.80 (1.0594)	2660 (1096)
Averaging	21.03 (0.5740)	4566 (1461)

6.6 Summary

This chapter treats a variant of the multi-robot task allocation problem where stochastic tasks arrive continuously, and the system must determine how to bundle tasks in order to make best use of synergies between tasks. First, we proposed a basic model to understand the foundations of bundling in this setting. Then, from empirical studies, we explored how the model changes as a function of bundle size, team size, task distribution method, and degree of synchronization. Based on this qualitative study, we proposed a set of simple bundling policies to optimize the system cost or the timespan. It is shown that bundling outperforms no bundling. Also, the policies are able to deal with uncertainties in the task profile, such as probabilistic task arrivals or non-i.i.d. task locations and arrival intervals.

7. CONCLUSION AND FUTURE WORK

In this dissertation, we solved MRTA problems subject to resource constraints, operating in dynamic environments, dealing with the risk arising from uncertainty, and serving online tasks. The results include: (i) polynomial-time task allocation algorithms considering shared resources (e.g., physical space, communication bandwidth) where a performance guarantee is provided for the approximation algorithm, (ii) algorithms that help mitigate the weaknesses in centralized systems (e.g., expensive global communication and centralized computations) when task costs change during execution, (iii) algorithms that analyze the sensitivity of optimal assignments with respect to risk (using risk measures such as the Conditional Value-at-Risk), and (iv) task bundling policies where tasks are revealed sequentially over time.

In Chapter 3, we defined the task allocation problems with resource contention and showed the complexity of them where problems with general and convex penalization functions are NP-hard and the problem with linear functions is in P. We developed an exact algorithm for general problems and two polynomial-time algorithms for the problems with convex and linear penalty functions. The algorithms produce an optimal or a high-quality approximation solution depending on the hardness of the problems. We applied the algorithms for a multi-robot transportation problem where narrow roads are shared by the robots, and the penalization function models traffic congestions precisely.

In Chapter 4, we proposed a region-based cost representation that incorporates uncertainty in costs. A finite region prescribes the possible changes of costs that might occur while robots are operating. We employed a sensitivity analysis of linear programming and showed that it reduces assignment re-computations when costs change. Also, we proposed three methods that remedy the weaknesses of the centralized structure in multi-robot systems by reducing the re-computations and global communication. We also showed that modeling interrelationships among costs makes cost regions tighter, thus we could have better predictions for cost changes.

Chapter 5 considered multi-robot task allocation under a probabilistic model of cost. We used random variables for costs to incorporate uncertainty and interdependency. Even without the assumption of independent costs, we showed that the costs chosen in an assignment are independent. We characterized the distributions by their expected values and CVaRs. We formulated a parameterized assignment problem where a risk preference determines the importance between the characterizations. Also, we showed a problem class that is indifferent to a particular value of the risk preference. For the risk preference dependent problems, we provided algorithms for analyzing the sensitivity of an optimal assignment with respect to the risk preference.

Chapter 6 treated an online variant of task allocation where stochastic and synergistic tasks arrive sequentially. The system must determine how many tasks the robots bundle to find the optimal point trading off between the system cost (e.g., fuel spent) and the timespan of tasks (e.g., the time from the task insertion to completion). We investigated a basic model to understand the foundations of bundling. We performed empirical studies and explored how the basic model changes depending on the bundle size, task distribution method, and the degree of synchronization. This qualitative study enabled us to propose a set of bundling policies

For future work, we would like to extend some of the work done and explore interesting questions in online task allocation. In Chapter 3, the algorithms use the centralized approach where a central unit computes an optimal assignment and distributes the assignment to other robots. Thus their use could be restrictive if central computation and global communication are not possible or the expense of central computation and global communication is prohibitive. We are interested in developing decentralized versions of the algorithms along with modeling contention on other types of (e.g., non-physical) resources. For Chapter 5, we would like to investigate further what conditions (i.e., properties of problem domains) make a task allocation problem risk indifferent. Chapter 6 brings interesting open questions in serving online tasks while optimizing the steady-state performance of the system. We plan to further study the strategies to improve the performance of bundled task execution. There are several directions for improvement. For example, robots may swap the tasks in their bundles for additional refinements. Preemptions of bundle executions may be useful so that some robots can stop working if they exceed the expected execution time for their bundles. We also wish to extend our study to tasks with negative synergies, no synergy, and (non-monotonic) complex synergies. Moreover, we are interested in other variants of tasks, such as tasks with deadline constraints or tasks that could be abandoned or rejected.

One of the findings in this dissertation study is that we identified probably not all, but some common and important interrelationships and uncertainties in robots, tasks, and environments that were able to be incorporated into the model of cost. But more importantly, we expanded the classical MRTA approaches in various directions. These expansions attempt to synthesize the decomposed components in task planning described in Fig. 1.1. First, robot interactions traditionally treated in the motion planning and execution time are modeled and embedded in the task allocation mechanism. Second, some portion of task assessment is combined with task allocation to model uncertainties. As a result, the cost models, which is an input to task allocation, become richer than the conventional scalar costs. Last, task allocation is extended for continuing operations that are uncertain.

Still, these efforts are done in the classical framework of task planning (e.g., Fig. 1.1). For a long-term goal, we want to develop a unified online task planning framework where the decomposed steps in task planning are combined so that planning can flow seamlessly without loosing useful information. Also, we want to not only preserve the richness of the information regarding task planning but also achieve tractability. A combination of efficient optimization methods and expressive models in artificial intelligence (e.g., partially-observable Markov decision process, linear temporal logic based planning, game theory) would achieve the both objectives.

REFERENCES

[1] Jose Acevedo, Begona Arrue, Ivan Maza, and Anibal Ollero. A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4735–4740, 2014.

[2] Javier Alonso-Mora, Martin Rufli, Roland Siegwart, and Paul Beardsley. Collision avoidance for multiple agents with joint utility maximization. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2833–2838, 2013.

[3] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, pages 1495–1496, 2014.

[4] Kurt Anstreicher. Linear programming in $o([\frac{n^3}{\text{ln}n}]l)$ operations. *SIAM Journal on Optimization*, 9(4):803–812, 1999.

[5] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9:203–228, 1999.

[6] Nuzhet Atay and Burchan Bayazit. Mixed-integer linear programming solution to multi-robot task allocation problem. Technical Report 2006-54, Washington University in St. Louis, 2006.

[7] Charles Audet, Pierre Hansen, Brigitte Jaumard, and Gilles Savard. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, 87:131–152, 2000.

[8] Alexander Bahr, John Leonard, and Maurice Fallon. Cooperative localization for autonomous underwater vehicles. *International Journal of Robotics Research*, 28(6):714–728, 2009.

[9] Ross Baldick. A unified approach to polynomially solvable cases of integer "non-separable" quadratic optimization. *Discrete Applied Mathematics*, 61(3):195–212, 1995.

[10] Michel Balinski. Signature methods for the assignment problem. *Operations Research*, 33(3):527–536, 1985.

[11] Mokhtar Bazaraa, John Jarvis, and Hanif Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[12] Jillian Beardwood, John Halton, and John Michael Hammersley. The shortest path through many points. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 55, pages 299–327. Cambridge Univ Press, 1959.

[13] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34:209–219, 2006.

[14] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. Siam, 2001.

[15] Curt Alexander Bererton. *Multi-robot coordination and competition using mixed integer and linear programs*. PhD thesis, Carnegie Mellon University, 2004.

[16] Dimitri Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, 1981.

[17] Silvia Botelho and Rachid Alami. M+: a scheme for multirobot cooperation through negotiated task allocation and achievement. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1234–1239, 1999.

[18] Stephen Bradley, Arnoldo Hax, and Thomas Magnanti. Applied mathematical programming. 1977.

[19] Richard Brualdi and Herbert Ryser. *Combinatorial matrix theory*, volume 39. Cambridge University Press, 1991.

[20] Francesco Bullo, Emilio Frazzoli, Marco Pavone, Ketan Savla, and Stephen Smith. Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, 99:1482–1504, 2011.

[21] Rainer Burkard and Eranda Cela. *Linear assignment problems and extensions*. Springer, 1999.

[22] Paul Camion. Characterization of totally unimodular matrices. *Proceedings of American Mathematical Society*, 16:1068–1073, 1965.

[23] Jian Chen, Dong Sun, Jie Yang, and Haoyao Chen. Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme. *International Journal of Robotics Research*, 29(6):727–747, 2010.

[24] Steve Chien, Anthony Barrett, Tara Estlin, and Gregg Rabideau. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of the International Conference on Autonomous Agents*, pages 100–101, 2000.

[25] Han-Lim Choi, Luc Brunet, and Jonathan How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

[26] Peter Corke. *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[27] Daniel Dadush, Chris Peikert, and Santosh Vempala. Enumerative lattice algorithms in any norm via m-ellipsoid coverings. In *Proceedings of IEEE Annual Symposium on Foundations of Computer Science*, pages 580–589, 2011.

[28] Torbjorn Dahl, Maja Matarić, and Gaurav Sukhatme. Multi-robot task-allocation through vacancy chains. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2293–2298, 2003.

[29] Vishnu Desaraju and Jonathan How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[30] Tamal Dey, Anil Hirani, and Bala Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM Journal on Computing*, 40(4):1026–1044, 2011.

[31] Jan Dhaene, Steven Vanduffel, MJ Goovaerts, Rob Kaas, Qihe Tang, and David Vyncke. Risk measures and comonotonicity: a review. *Stochastic models*, 22:573–606, 2006.

[32] Bernadine Dias and Tuomas Sandholm. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, 2004.

[33] Bernardine Dias and Anthony Stentz. A market approach to multirobot coordination. Technical Report CMU-RI -TR-01-26, Carnegie Mellon University, 2000.

[34] Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94:1257–1270, 2006.

[35] Timon Du, Eldon Li, and An-Pin Chang. Mobile agents in distributed network management. *Communications of the ACM*, 46:127–132, 2003.

[36] Carlo Filippi. A fresh view on the tolerance approach to sensitivity analysis in linear programming. *European Journal of Operational Research*, 167:1–19, 2005.

[37] Jim Florwick, Jim Whiteaker, Alan Amrod, and Jake Woodhams. Wireless lan design guide for high density client environments in higher education. In *Design guide*. Cisco Systems, 2013.

[38] Thomas Gal. *Postoptimal analyses parametric programming and related topics*. McGraw-Hill, 1979.

[39] Tomas Gal, Hermann-Josef Kruse, and Peter Zörnig. *Survey of solved and open problems in the degeneracy phenomenon*. Springer, 1988.

[40] Tomas Gal and Josef Nedoma. Multiparametric linear programming. *Management Science*, 18:406–422, 1972.

[41] Brian Gerkey and Maja Matarić. Murdoch: Publish/subscribe task allocation for heterogeneous agents. In *Proceedings of the International Conference on Autonomous Agents*, pages 203–204, 2000.

[42] Brian Gerkey and Maja Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics*, 18:758–768, 2002.

[43] Brian Gerkey and Maja Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23:939–954, September 2004.

[44] Stefano Giordani, Marin Lujak, and Francesco Martinelli. A distributed algorithm for the multi-robot task allocation problem. In *Trends in Applied Intelligent Systems*, pages 721–730. Springer, 2010.

[45] Dani Goldberg. *Evaluating the dynamics of agent-environment interaction*. PhD thesis, University of Southern California, 2001.

[46] Dani Goldberg and Maja Matarić. Interference as a Tool for Designing and Evaluating Multi-Robot Controllers. In *Proceedings of AAAI National Conference on Artificial Intelligence*, pages 637–642, 1997.

[47] Clóvis Gonzaga. *An algorithm for solving linear programming problems in $O(n^3L)$ operations*. Springer, 1989.

[48] Google. The Google Directions API. https://developers.google.com/maps/documentation/directions/, 2013.

[49] Google. The Google Directions API.
https://developers.google.com/maps/documentation/directions/, 2016.

[50] Harvey Greenberg. An analysis of degeneracy. *Naval Research Logistics Quarterly*, 33:635–655, 1986.

[51] José Guerrero and Gabriel Oliver. Physical interference impact in multi-robot task allocation auction methods. In *Proceedings of IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, pages 19–24, 2006.

[52] Liang He and Jan van den Berg. Meso-scale planning for multi-agent navigation. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2839–2844, 2013.

[53] Bradford Heap. *Sequential Single-Cluster Auctions for Multi-Robot Task Allocation*. PhD thesis, The University of New South Wales, 2013.

[54] Bradford Heap and Maurice Pagnucco. Sequential single-cluster auctions for robot task allocation. In *Advances in Artificial Intelligence*, pages 412–421. 2011.

[55] Dorit Hochbaum and George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, 37(4):843–862, 1990.

[56] Sue Ann Hong and Geoffrey Gordon. Decomposition-based optimal market-based planning for multi-agent systems with shared resources. In *International Conference on Artificial Intelligence and Statistics*, volume 15, pages 351–360, 2011.

[57] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology, 1992.

[58] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[59] Richard Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[60] Young-Ho Kim and Dylan Shell. Distributed robotic sampling of non-homogeneous spatio-temporal fields via recursive geometric sub-division. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 557–562, 2014.

[61] Leonard Kleinrock. *Queuing systems*. Wiley, 1975.

[62] Sven Koenig, Craig Tovey, Xiaoming Zheng, and Ilgaz Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of International Joint Conference on Artificial intelligence*, pages 1359–1365, 2007.

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

[64] Pavlo Krokhmal and Panos Pardalos. Random assignment problems. *European Journal of Operational Research*, 194(1):1–17, 2009.

[65] Harold Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2(1-2):83–97, 1955.

[66] Akshat Kumar, Boi Faltings, and Adrian Petcu. Distributed constraint optimization with structured resource constraints. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 923–930, 2009.

[67] Jooyoung Lee and MooYoung Choi. Optimization by multicanonical annealing and the traveling salesman problem. *Physical Review E*, 50:R651, 1994.

[68] Hendrik Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, pages 538–548, 1983.

[69] Sven Leyffer and Ashutosh Mahajan. Nonlinear constrained optimization: methods and software. *Argonee National Laboratory, Argonne, Illinois*, 2010.

[70] Chi-Jen Lin and Ue-Pyng Wen. Sensitivity analysis of objective function coefficients of the assignment problem. *Asia-Pacific Journal of Operational Research*, 24:203–221, 2007.

[71] Lantao Liu and Dylan Shell. Assessing optimal assignment under uncertainty: An interval-based algorithm. *International Journal of Robotics Research*, 30(7):936–953, 2011.

[72] Lantao Liu and Dylan Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33:291–307, 2012.

[73] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Multi-robot assignment algorithm for tasks with set precedence constraints. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2526–2533, 2011.

[74] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3007–3013, 2013.

[75] Andriy Andreev-Antti Kanto-Pekka Malo. On closed-form calculation of cvar. 2005.

[76] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7:77–91, 1952.

[77] Toshihiro Matsui, Hiroshi Matsuo, Marius Silaghi, Katsutoshi Hirayama, and Makoto Yokoo. Resource constrained distributed constraint optimization with virtual variables. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 120–125, 2008.

[78] Reshef Meir, Yiling Chen, and Michal Feldman. Efficient parking allocation as online bipartite matching with posted prices. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 303–310, 2013.

[79] Ayorkor Mills-Tettey, Anthony Stentz, and Bernardine Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. 2007.

[80] Mosek. The mosek optimization software version 6. *Online at http://www.mosek.com*, 2009.

[81] Robin Murphy, Karen Dreger, Sean Newsome, Jesse Rodocker, Brian Slaughter, Richard Smith, Eric Steimle, Tetsuya Kimura, Kenichi Makabe, Kazuyuki Kon, et al. Marine heterogeneous multirobot systems at the great eastern japan tsunami recovery. *Journal of Field Robotics*, 29(5):819–831, 2012.

[82] Katta Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16:682–687, 1968.

[83] Changjoo Nam and Dylan Shell. Assignment algorithms for modeling resource contention in multi-robot task-allocation. *IEEE Transactions on Automation Science and Engineering*, 12:889–900, 2015.

[84] Changjoo Nam and Dylan Shell. When to do your own thing: Analysis of cost uncertainties in multi-robot task allocation at run-time. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1249–1254, 2015.

[85] Yurii Nesterov, Arkadii Nemirovskii, and Yinyu Ye. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994.

[86] Gordon Newell. Nonlinear effects in the dynamics of car following. *Operations Research*, 9(2):209–229, 1961.

[87] Evdokia Nikolova and Nicolas Stier-Moses. A mean-risk model for the stochastic traffic assignment problem. *Operations Research*, 62:366–382, 2014.

[88] Michael Otte and Nikolaus Correll. The any-com approach to multi-robot coordination. In *Proceedings of the ICRA Workshop on Network Science and Systems Issues in Multi-Robot Autonomy*, 2010.

[89] James Parker, Alessandro Farinelli, and Maria Gini. Decentralized allocation of tasks with costs changing over time. *IJCAI Workshop on Synergies between Multi-agent Systems, Machine Learning and Complex Systems*, pages 62–73, 2015.

[90] Lynne Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics*, 14:220–240, 1998.

[91] Fabio Pasqualetti, Antonio Franchi, and Francesco Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *IEEE Transactions on Robotics*, 28(3):592–606, 2012.

[92] Giovanni Pini, Arne Brutschy, Mauro Birattari, and Marco Dorigo. Interference reduction through task partitioning in a robotic swarm. In *Proceedings of International Conference on Informatics in Control, Automation and Robotics*, pages 52–59, 2009.

[93] Sameera Ponda, Luke Johnson, and Jonathan How. Distributed chance-constrained task allocation for autonomous multi-agent teams. In *Proceedings of the American Control Conference*, pages 4528–4533, 2012.

[94] Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[95] Tim Roughgarden. Routing games. *Algorithmic game theory*, 18, 2007.

[96] Sanem Sariel, Tucker Balch, and Nadia Erdogan. Incremental multi-robot task selection for resource constrained and interrelated tasks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 2314–2319, 2007.

[97] Dylan Shell and Maja Matarić. On foraging strategies for large-scale multi-robot systems. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 2717–2723, 2006.

[98] Wei-Min Shen and Behnam Salemi. Distributed and dynamic task reallocation in robot organizations. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1019–1024, 2002.

[99] Pedro Shiroma and Maria Fernando Montenegro Campos. Comutar: A framework for multi-robot coordination and task allocation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 4817–4824, 2009.

[100] David Stein. An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*, 3:89–101, 1978.

[101] Fang Tang and Lynne Parker. A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation. In *Proceedings*

of IEEE International Conference on Robotics and Automation, pages 3351–3358, 2007.

[102] Ekunda Ulungu and Jacques Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:149–165, 1995.

[103] Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. Optimal online assignment with forecasts. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 109–118, 2010.

[104] James Ward and Richard Wendell. Approaches to sensitivity analysis in linear programming. *Annals of Operations Research*, 27:3–38, 1990.

[105] Barry Werger and Maja Matarić. Broadcast of local eligibility for multi-target observation. *Distributed Autonomous Robotic Systems 4*, pages 347–356, 2001.

[106] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–19, 2008.

[107] Jingjin Yu and Steven LaValle. Planning optimal paths for multiple robots on graphs. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3612–3617, 2013.

[108] Ernst Zermelo. Über das navigationsproblem bei ruhender oder veränderlicher windverteilung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 11(2):114–124, 1931.

[109] Lei-Hong Zhang, Wei Yang, and Li-Zhi Liao. On an efficient implementation of the face algorithm for linear programming. *Journal of Computational Mathematics*, 31(4):335–354, 2013.

[110] Yu Zhang and Lynne Parker. Considering inter-task resource constraints in task allocation. *Autonomous Agents and Multi-Agent Systems*, 26:389–419, 2013.

[111] Xiaoming Zheng, Sven Koenig, and Craig Tovey. Improving sequential single-item auctions. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 2238–2244, 2006.

APPENDIX A

A TRANSFORMATION OF A NONSEPARABLE PROBLEM TO A SEPARABLE PROBLEM

Here, we show the transformation of a nonseparable C-type problem to a separable problem, and show how to check the totally unimodularity of the transformed problem.

Suppose that $n = m = 3$, $p_{ij} = 2, \forall\{i, j\}$, and a convex quadratic penalization function $Q(\cdot)$ in (3.1) is

$$\begin{aligned} &Q(x_{111}, x_{112}, \ldots, x_{331}, x_{332}) \\ &= (x_{111} + x_{121} + x_{131} + x_{211} + x_{221} + x_{231} \\ &\quad + x_{311} + x_{321} + x_{331})^2 + (x_{112} + x_{122} + x_{132} \\ &\quad + x_{212} + x_{222} + x_{232} + x_{312} + x_{322} + x_{332})^2 \end{aligned} \tag{A.1}$$

which is (3.14) where $\beta_{\mathrm{C}} = 1$ and $\beta'_{\mathrm{C}} = \beta''_{\mathrm{C}} = 0$. (A.1) can be written as

$$Q(\cdot) = {y_1}^2 + {y_2}^2 \tag{A.2}$$

where

$$\begin{aligned} y_1 &= x_{111} + x_{121} + \ldots + x_{321} + x_{331}, \\ y_2 &= x_{112} + x_{122} + \ldots + x_{322} + x_{332}. \end{aligned}$$

Thus, additional constraints

$$x_{111} + x_{121} + \ldots + x_{321} + x_{331} - y_1 = 0,$$

$$x_{112} + x_{122} + \ldots + x_{322} + x_{332} - y_2 = 0$$

are added to (3.2)–(3.5). Therefore, A_{N} in (3.12) is

$$A_{\text{N}} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix},$$

and this is TU by definition. Since a TU A_{N} does not always guarantee a TU A_{SP}, A_{SP} also has to be checked. A_{SP} is not TU because at least one of its submatrix has a determinant other than -1, 0, or 1 (e.g., $\det([1\ 1\ 0; 1\ 0\ 1; 0\ 1\ 1]) = -2$). Therefore, this C-type problem instance does not belong to the polynomial-time solvable class of the C-type problem.

APPENDIX B

UNDERSTANDING DEGENERACY IN LP

One easy way to understand degeneracy in LP is using a polytope defined by constraints of an optimization problem. In nondegenerate cases, an extreme point of a polytope corresponds to one feasible solution. In degenerate cases, one extreme point corresponds to many different degenerate solutions. It is worth noting that the stalling and cycling problems in the Simplex method are caused by pivoting between the multiple degenerate solutions on the same extreme point. See [50, 39] for more details.

APPENDIX C

A HARDNESS PROOF FOR THE NONLINEAR CONVEX COST BOUNDARY

Line 6 of Alg. 4 is a linear programming problem. The objective function is linear, and the linearly of the constraint set depends on which type of cost boundary is used. If the boundary is nonlinear convex, the problem becomes the optimization of a linear objective function subject to a nonlinear convex constraint set (LONC).

In this appendix, we prove that LONC is in P to show that Alg. 4 for LONC still runs in polynomial time. We show a polynomial-time reduction from LONC to the optimization of a nonlinear convex objective function subject to a linear convex set (NOLC) to prove LONC is in P.

Theorem C.17 LONC is in P.

Claim. If LONC $\leq_{\mathrm{P}}$ NOLC, LONC is in P since NOLC is proven to be in P [14].

Proof. In line 6 of Alg. 4, the objective function of the linear programming is separable. If both the objective function and the set of constraints are separable, LONC is easily transformed to NOLC in polynomial time by variable substitution (an example follows after this proof). If the constraint set is represented by nonseparable functions, some known methods in [18, Table 13.1] can transform nonseparable functions to separable in polynomial time. Therefore, LONC $\leq_{\mathrm{P}}$ NOLC. □

An example of LONC is

$$\max x_1 + x_3 - x_4$$

subject to

$$x_1^2 + x_3^4 \geq 1$$

$$x_2^2 + x_4^2 \geq 1.$$

This can be transformed to

$$\max \sqrt{y_1} + \sqrt[4]{y_3} - \sqrt{y_4}$$

subject to

$$y_1 + y_3 \geq 1$$

$$y_2 + y_4 \geq 1$$

by substituting $y_i = x_i^2$ for $i = 1, \cdots, 4$. Now the transformed objective function is convex and the constraints are linear, which is NOLC.

APPENDIX D

A RANDOMIZED HEURISTIC HP ALGORITHM

We implement a randomized heuristic algorithm for the HP problem. The algorithm receives the input $\mathbf{X}_i$ where $|\mathbf{X}_i| = x_i$. It generates a list $\mathbf{X}'_i$ of two elements: the initial robot location and a randomly chosen visit location from $\mathbf{X}_i$ (line 1). The chosen location is removed from $\mathbf{X}_i$ (line 2). We consider the insertion positions in $\mathbf{X}'_i$: between the two elements and after the last element (we do not consider the foremost position before the elements). In general, there are $|\mathbf{X}'_i|$ insertion positions in $\mathbf{X}'_i$. Let P_l be the insertion position for $l = 1, \cdots, |\mathbf{X}'_i|$.

The algorithm involves the following procedure for all visit locations in $\mathbf{X}_i$. From the first location, (i) inserting the chosen visit location at each insertion position and generating multiple paths (line 6), and (ii) choosing the minimum length path among the paths generated from i) (line 8). $\mathbf{X}'_i$ increases by one in each iteration by inserting one visit location. Once all visit locations are inserted to $\mathbf{X}'_i$ (at line 10), the path of x^i visit locations has a reasonably small distance but still not optimal (or near-optimal). Now the algorithm randomly chooses one visit location T_b and removes it from $\mathbf{X}'_i$ (line 12). Then the algorithm runs (i) and (ii) with T_b and $\mathbf{X}'_i$ (lines 13–16). If the resulting path is better than the previous one, the algorithm updates the path (line 18). This improvement repeats for γx_i iterations, where $\gamma \in \mathbb{Z}^+$. The resulting Hamiltonian path would not be optimal but near-optimal. For better results (but probably a longer running time), the stopping point could be the time when the path distance converges.

Algorithm 11 RANDHP

Input: $\mathbf{X}_i$, a bundle of x_i tasks

Output: $\mathbf{X}'_i$, a reordered $\mathbf{X}_i$ forming a Hamiltonian path

```
1  $\mathbf{P} = \emptyset$, $s = 0$
2  $\mathbf{X}'_i = \{l(R_i), l(T_a)\}$//$T_a$ is randomly chosen from $\mathbf{X}_i$
3  $\mathbf{X}_i = \mathbf{X}_i \setminus \mathbf{X}'_i$
4  for each $T_j \in \mathbf{X}_i$
5    for each $P_k \in \mathbf{X}'_i$
6      $\mathbf{P} \leftarrow$ INSERT($T_j, P_k$)//insert $T_j$ in $P_k$ and add to $\mathbf{P}$
7    end for
8    $\mathbf{X}'_i =$ MINPATH($\mathbf{P}$)//find the minimum length path
9    $\mathbf{P} = \emptyset$
10 end for
11 while $s < \gamma x_i$
12   $\mathbf{X}'_i = \mathbf{X}'_i \setminus T_b$//$T_b$ is randomly chosen from $\mathbf{X}'_i$
13   for each $P_k \in \mathbf{X}'_i$
14     $\mathbf{P} \leftarrow$ INSERT($T_b, P_k$)//insert $T_b$ in $P_k$ and add to $\mathbf{P}$
15   end for
16   $\mathbf{X}''_i =$ MINPATH($\mathbf{P}$)//find the minimum length path
17   if DIST($\mathbf{X}'_i$)>DIST($\mathbf{X}''_i$)//compare distances of paths
18     $\mathbf{X}'_i = \mathbf{X}''_i$//if the randomly modified path $\mathbf{X}''_i$ is
               //better than the previous one, update $\mathbf{X}'_i$
19   end if
20   $\mathbf{P} = \emptyset$
21 end while
22 return $\mathbf{X}'_i$
```