Multiple Robot Graph Exploration

Hui Wang

Technical Report CSE-2007-06

November 2007

Department of Computer Science and Engineering

4700 Keele Street Toronto, Ontario M3J 1P3 Canada

MULTIPLE ROBOT GRAPH EXPLORATION

HUI WANG

TECHNICAL REPORT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO
NOVEMBER 2007

Abstract

This report investigates the problem of exploration and mapping in an embedded graph-like world. Graph-like worlds provide a useful theoretical model within which to explore fundamental limits to exploration and mapping. It is demonstrated that a collection of identical robots, each equipped with its own unique marker can explore and map an unknown graph-like environment. Developing such an algorithm addresses fundamental issues related to multiple-robot exploration, i.e., location disambiguation, merging partial world representations obtained by multiple robots, partitioning the exploration task, and rendezvous scheduling.

Table of Contents

1 Introduction

Mobile robots have shown significant promise for remote exploration, going places that are too distant, too dangerous, or simply too costly to allow human access (Figure 1.1). If robots are to operate autonomously in extreme environments such as undersea, underground, or on the surfaces of other planets, they must be capable of building maps and navigating reliably according to these maps [39]. Even in benign and simpler environments such as the interiors of buildings, accurate mapping of the environment is important. Without a map, many robotic tasks are difficult or even impossible. For example, asking a robot to 'go to my office' requires a sufficiently rich representation to encode the concept of places and paths between them.

Maps typically serve as the basis for motion planning, but maps often have value in their own right [39]. For example, in July of 2002, nine miners in the Quecreek Mine in Sommerset, Pennsylvania, were trapped underground for three and a half days after accidentally drilling into a nearby abandoned mine. A subsequent investigation attributed the cause of the accident to the use of inaccurate maps [39, 29]. Since the accident, mobile robots and more advanced approaches have been investigated as a possible technology for acquiring accurate maps of abandoned mines. One such robot, shown in Figure 1.1(b), is capable of building 3D reconstructions of the interior of abandoned mines [57].

Acquiring maps with mobile robots is a challenging problem for a number of reasons [55]. To acquire a map, robots must possess sensors that enable it to perceive the external world. However, all sensors are subject to errors and range limitations, and errors

(a) Undersea robot (b) Underground robot (c) Robot on Mars

Figure 1.1: Various applications of robot exploration and mapping. Courtesy of M. Montemerlo ([39]).

accumulate over time. This is illustrated in Figure 1.2, which shows an example path of a mobile robot in a given map of the environment. In the example a small rotational error in the robot's initial orientation leads to a significant error between the map and laser measurements. Possible changes in the environment is another challenging problem. For example, facing a closed door that was previously modeled as open introduces another way in which the seemingly inconsistent sensor measures can be interpreted. A critical challenge arises from the fact that a robot must represent itself within the map as it is being constructed. This requires the robot to estimate its pose in the map (localization). The mapping problem is therefore like a 'chicken and egg' problem: constructing the map requires the solution to localization, and solving localization requires a solution to mapping. In the absence of both an initial map and exact pose information, the problem is hard.

The combined problem has been coined *SLAM*, which is short for *Simultaneous Localization and Mapping*. In the majority of *SLAM* approaches the spatial description of the environment is represented through a metric map that captures the geometric properties

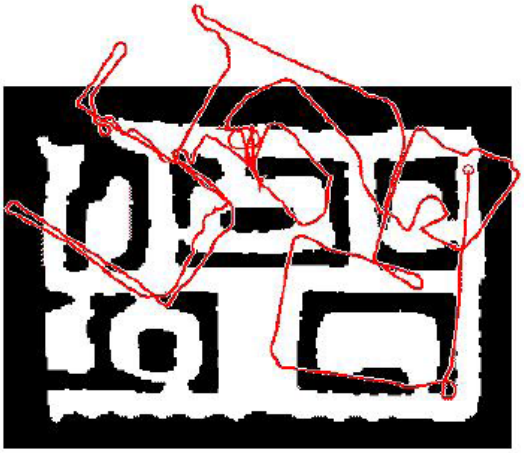


Figure 1.2: Small odometry (or control) errors can have large effects on later position estimates (robot started in top-right). Courtesy of S. Thrun ([55]).

of the environment [55]. A common approach to solving the SLAM problem is to cast the problem within a probabilistic framework. Due to the strong reliance on sensor measurements, those probabilistic approaches are challenged by the nature of the measurement noise, e.g., the solution fails when measurement error accumulates over a certain level and that level must be known or approximated *a priori*.

An alternative to a metric-based representation is a representation based on a topological or graph-like representation. A graph-like world represents the minimal information that a robot must be able to represent in order to distinguish one place from another. Dudek et al. [14, 15] developed a non-probabilistic SLAM algorithm for graph-like worlds. In this work the world is modeled as a graph embedding, i.e., a graph in which there exists an ordering of edges incident upon each vertex. The model supposes the existence of a unique marker (also referred to as *beacon* or *pebble* in the literature) that can be used to help disambiguate locations in the environment (vertices in the graph-like map). The approach assumes no distance or orientation metric. It has been shown that graph-like worlds can be fully explored and mapped by a single robot equipped with a unique

marker [14, 15].

Dudek et al.'s single robot exploration algorithm in [14, 15] is a deterministic and provably correct *SLAM* solution to the graph exploration problem. In later work Dudek et al. provided an informal extension of the above algorithm to the case of multiple robots [20]. This extension assumes the same environmental representation as described in [14, 15] and populates the world with two or more robots each of which is equipped with its own marker. In this extension the robots can only communicate when they are in the same vertex. A multiple robots system is expected to provide performance that is faster and more robust than a single robot system but the algorithm also faces challenges including how to merge the partial representations obtained by individual robots, how to partition the task between the robots, and rendezvous scheduling. The algorithm sketched in [20] suggests several interesting extensions and variations and leaves several practical problems to be addressed. It is worthwhile exploring these core techniques, with an aim to develop a formal specification of the multiple robot exploration problem and, based on the formal specification, investigate possible enhancements to the algorithm.

Problem statement and contribution

This report extends Dudek et al.'s model to the case of multiple robots, addressing the core techniques necessary for the problem of multiple robot exploration, i.e., location disambiguation, merging partial representations obtained by the individual robots, rendezvous scheduling, and partitioning of the exploration task.

This report also explores a number of enhancements and extensions that can be made to both Dudek et al.'s single robot exploration algorithm [14, 15] and the multiple robot exploration algorithm [20]. The enhancements address some of the core techniques involved in multiple robot exploration.

Structure of the report

The remainder of this report is organized as follows. Chapter 2 defines formally the problem addressed in this report and surveys key techniques and related work in the field of robotic mapping and exploration, with an extensive description of the deterministic topological approach that is closely related to the work in this report. Open problems are also identified. Chapter 3 defines formally the model of the work in this report, develops and proves correct a basic multiple robot exploration algorithm and develops an evaluation mechanism for multiple robot exploration algorithms. Chapter 4 explores a number of enhancements that can be made to the multiple robot exploration algorithm. Chapter 5 concludes the work and presents a discussion of possible directions for future work.

2 Previous Work

This chapter begins by formally defining the problem to be addressed, and then reviews approaches in the literature to the problem. The chapter concludes with an overview of some of the open problems in single and multiple-robot exploration using a topological representation.

2.1 Problem definition

The problem addressed in this report can be formally summarized as follows: Given a specific environment that can be modeled as an embedded graph, formulate a series of plans for a group of multiple robots, so that after carrying out the actions specified by the plans, the robots will have constructed a topological map that is isomorphic to the underlying world being explored. In the development of such an algorithm a number of critical problems must be addressed, including

- Given that inter-robot communication is limited to vertices, how should rendezvous be scheduled so that the robots can explore the environment efficiently?
- Can the robots perform their exploration in a strategic way, taking advantage of the fact that exploration becomes easier as more of the world is explored?
- Can local neighbor information be exploited so that location disambiguation can be facilitated?

- How should parallelism be exploited during the merge process? How should merge be scheduled for the case of a large group of robots?
- How should the robots communicate when they meet opportunistically so that exploration and merge task can be facilitated?
- How should the robots divide up the world being explored in order to explore the world efficiently?

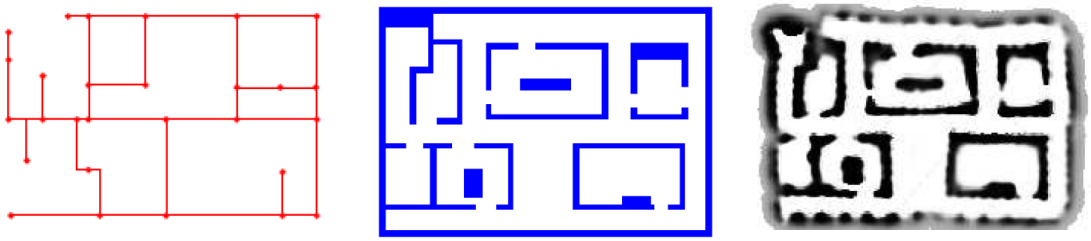
2.2 Robotic Mapping

Robotic exploration and mapping addresses the problem of acquiring spatial models (a 'map') of physical environments autonomously using mobile robots. This problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robot systems.

2.2.1 Mapping, localization and spatial representations

Formally, robot localization is the problem of determining the pose (location and orientation) of a robot relative to a given map of the environment [56]. The robot mapping problem therefore addresses two interrelated problems in robotics, namely, *localization*, which is the problem of determining a robot's pose in the growing map ('where am I in the world?'), and *mapping*, which is the problem of constructing a spatial representation (map) of the environment ('what does the world look like?'). Maps themselves can be categorized as being either *metric* or *topological*, and metric maps can be further subdivided into *feature based* or *location based* representations [56] (Figure 2.1).

Metric representation Metric maps adopt a metric representation of space, which captures the geometric properties (e.g., coordinates in a Cartesian representation) of



(a) Topological (graph-like) map

(b) Feature (landmark) based metric map

(c) Occupancy Grid map

Figure 2.1: Different spatial representations. Courtesy of S. Thrun [56].

the environment. Metric representations can be further divided into *feature (landmark) based* maps and *location based* maps. In feature (landmark) based maps, the world is represented as a set of spatially located features (landmarks), each with an associated coordinate in the metric space. In location based maps, the world is represented as a set of locations. A representative example of a location based map is the occupancy grid map [23], in which the space is represented as a fine-grained grid defined over the continuous space of locations. The main advantage of the feature-based map is its compactness, which makes it suitable for operating in large environments. The primary drawback of a feature based map is the need for some feature extraction mechanisms, which usually assume prior knowledge of some structure in the environment [56]. The main advantage of a location based map (e.g., occupancy grid map) representation is that it can be used to represent unstructured environments. In addition, operating with location based maps does not require the use of feature extraction mechanism, making the overall approach more robust. The disadvantages of location based maps (e.g., occupancy grid maps) include the fact that cells in the maps are considered independent so the representation does not maintain dependencies among the cells, and the large storage requirements

associated with the them [55].

Topological representation Topological maps describe the connectivity of different places. Environments in topological maps are represented as a list of significant places (vertices) that are connected via arcs (edges). One advantage of the representation is the abstract view of environment and the consequent low space complexity [55]. Another advantage of the model is that map learning can be approached as a graph theoretic problem, making it feasible to investigate general issues related to robot exploration within this representation. In practice, metric maps are finer grained than topological representations.

2.2.2 Probabilistic and deterministic representations

It is possible to build a map representation (either metric or topological) that is either deterministic or probabilistic. Whereas deterministic schemes usually resort to the use of additional tools (e.g., markers, threads) for disambiguation purpose, a probabilistic representation uses probabilistic concepts to explicitly represent and manipulate spatial uncertainty. Early work with probabilistic representations includes [21, 22, 4, 38, 40, 53, 52]. Since the 1990's, with the introduction of powerful statistical frameworks for simultaneously solving the mapping problem and the induced problem of localizing the robot relative to its growing map, the field of robot mapping has been dominated by probabilistic techniques [55]. The reason for the popularity of probabilistic techniques stems from the fact that robot mapping is characterized by uncertainty and sensor noise by pose. Probabilistic algorithms approach the problem by explicitly modeling these different sources of noise and their effects on the measurements and the map models.

In robotic mapping and localization the task is usually modeled as inferring a state quantity x (map or robot pose), based on some data d (measurement or control). Within

a probabilistic framework, this is represented as $p(x|d)$, often referred as *posterior probability*, or *belief* [55].

2.3 Metric approaches to mapping and localization

Metric approaches to mapping and localization adopt a metric representation of space in order to capture the geometric properties of the environment. Such approaches typically use probabilistic concepts to explicitly represent and manipulate spatial uncertainty. In *SLAM*, which is the problem of estimating both the mapping and localization problem at the same time [56], the task is modeled as estimating a posterior probability distribution over all possible states, i.e., all possible maps and all possible robot poses, given the controls and sensor readings accumulated by the robot. This distribution is called the *SLAM posterior* [39]. The Kalman filter [56, 34] and the particle filter [56, 27] are important techniques to approximate the SLAM posterior.

Dudek et al. [12] summarized reasons and situations that make the representation and construction of maps based on metric information alone problematic or inappropriate, including: Long-term goals are often expressed in terms of semantic tokens or places, rather than specific coordinates (i.e. going to a valley, a route or a room rather than a specific coordinate); Absolute coordinate systems are typically very difficult to accurately maintain at every scale; Complete metric representations may involve very large amounts of data; And changes in the environment and the correspondence between objects may be difficult to establish in a purely metric representation. For these and other reasons there is interest in the use of more abstract map representations. Motivated by the need for spatial representation other than the ones based on metric information, Kuipers et al. [37] proposed a four-level spatial semantic hierarchy, in which metric and topological representations are associated. The four levels, starting from the lowest, are

1. Sensorimotor (robot sensations and primitives actions).

2. Procedural (robot actions to accomplish place-finding and route-following tasks).

3. Topological (places and paths and their topological relations).

4. Metric (places and paths and their metric relations).

According to E. Davis [10], a topological map can be defined as a map including all fixed entities in the world such as distinguishable places and regions, linked by topological relations, e.g., connectivity. Such a map is often represented as a graph where vertices are places and edges denote their adjacency relations. Kuipers et al. proposed a topological model of the world [35, 36, 37]. In this model, a place is defined as a point which maximizes some distinctiveness measures (its *signature*) allowing it to be locally distinctive within its immediate neighborhood.

2.4 The deterministic SLAM approach of Dudek et al.

In order to avoid dealing with the problems encountered in making primitive measurements for the construction of metric maps, and in general with many of the low level tasks associated with robotics, Dudek et al. [14, 15] proposed a model based on a topological representation of the environment. This approach assumes that the world can be modeled as a collection of isolated locations of interest that are connected by featureless pathways. Specifically, the world is modeled as a graph embedding consisting of vertices, a set of edges between them and an ordering defined on all edges incident upon each vertex. A similar world model is also assumed in [16, 12, 48, 11]. The work in [14, 15] presents a deterministic approach to the SLAM problem, in which disambiguation is achieved through the use of a recognizable marker that can be put down and picked up by the robot.

There are several properties of the model in [14, 15] that are worth noting. First is the embedding assumed. With embedding, there exists an ordering defined on all edges incident upon each vertex. The ordering captures the relative direction (orientation) in

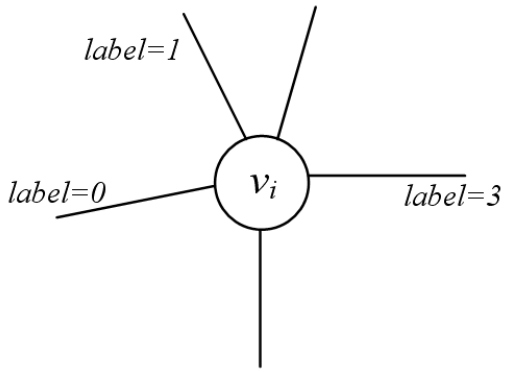


Figure 2.2: A vertex with fixed order (relative direction) of edges (exits) leave the vertex.

which edges (viewed as exits) leave a vertex (Figure 2.2). As long as one edge is identified, the labelling or ordering of the others can be determined. Without embedding, for a vertex having d incident edges, there would be $d!$ ways of labeling the edges, in terms of their relative positions. This embedding assumption, together with the relevant assumption that a robot can enumerate edges in a consistent way, greatly simplifies the problem. The power of embedding can be seen throughout the algorithm. For example, in [14, 15] a route (path) can be specified as a sequence of edge ordering (relative direction) with respect to the entry edge, e.g., 'take the 3rd exit on the right (with respect to the entry edge), upon arrival take the 2nd exit on the right'. Such a specification of route or movement would not be possible without an embedding (any other edges can be 'the 3rd right exit' with respect to the entry exit). Another property of the model is that this graph-like representation is minimalist. In the model, edges are completely featureless and vertices are featureless except for the paths to other vertices. In the notion of node *signature* proposed by Kuipers [36], the degree of a vertex is used as a specific instance of the signature function, and the marker is used to establish a 'temporary' unique signature of a vertex, making it distinctive relative to other vertices. No error-prone spatial metric

such as distance or orientation is required, i.e., the algorithm operates on a topological representation that is devoid of metric information. Techniques that are able to build an environmental map without such metric information can be viewed as assuming worst-case performance bounds for environments with noisy metric observation. In reality much more information is likely to be available at a given location than is assumed under this model. Such information would only help to make the signature more distinctive and therefore make the problem easier [36].

It has been shown in [14, 15] that such graph-like worlds can be fully explored and mapped by a single robot equipped with a unique marker. Compared with [36], the work can be viewed as introducing a different rehearsal procedure that includes portable markers, since the marker makes the place uniquely distinctive to other places. In contrast to the work on random walks of a graph [3], Dudek et al.'s approach visits vertices of a graph according to deterministic strategies. Related to the hierarchy proposed by Kuipers et al. [37], the work provided precise definitions of what corresponds to the sensorimotor, procedural and topological levels associated with the learning and navigation in a graph-like world, assuming no metric information is either sensed or stored.

The following sections present a detailed introduction to the model and exploration algorithm in [14, 15]. This model and algorithm form the basis of the algorithm developed within this report.

The Model

The world in [14, 15] is modeled as an augmented graph. The abstraction used serves as a lowest common denominator for real robotic systems, but is sufficient to represent location in a meaningful way. The goal of the robot's exploration is to build an augmented undirected graph that is isomorphic [30] to the finite world it has been assigned to explore. The robot's inputs are its sensations and it can interact with the world only through its

actions. The robot's actions and sensations are rather impoverished.

The World The following definitions follow those provided in [14]. The world is defined as an embedding of an undirected graph $G = (V, E)$ with a set of vertices $V = \{v_1, ..., v_n\}$ and a set of edges $E = \{(v_i, v_j)\}$. The world is assumed to have no multiple edges between two vertices and no edge incident twice at the same vertex. The definition of an edge is extended to allow for the explicit specification of the order of edges incident upon each vertex of the graph embedding. This ordering is obtained by enumerating the edges in a systematic (e.g. clockwise) manner from some standard starting direction. An edge $e_{i,j}$ incident upon v_i and v_j is assigned labels n and m, one for each of v_i and v_j respectively. n and m represent the ordering of the edge $e_{i,j}$ with respect to the consistent enumeration of edges at v_i and v_j respectively. The labels n and m can be considered as general directions, e.g., from vertex v_i then n'th exit takes edge $e_{i,j}$ to vertex v_j. A route (path) can be specified as a sequence of edge labels such that the entry edge at a vertex is always the reference edge and the successive labels specify the exit edges (e.g., take the $3rd$ edge on the right, then take the $2nd$ edge on the right etc).

Movement and Marker operation A robot can move from one vertex to another by traversing an edge (a *move*), it can pick up a marker that is located at the current vertex and it can put down a marker it holds at the current vertex (a *marker operation*). Assume the robot is at a single vertex v_i, having entered the vertex through edge $e_{l,i}$. In a single move, it leaves vertex v_i for v_j by traversing the edge $e_{i,j}$, which is located r edges after $e_{l,i}$ according to the edge order at vertex v_i (Figure 2.3). This can be formally expressed by the transition function

$$\delta(v_i, e_{l,i}, r) = v_j.$$

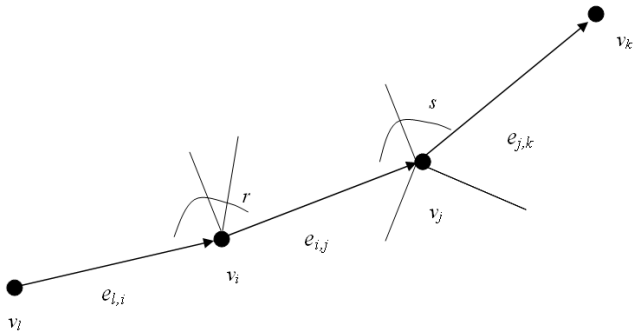


Figure 2.3: Transition Function.

The following property of the transition function is assumed: if $\delta(v_i, e_{l,i}, r) = v_j$ and $\delta(v_j, e_{i,j}, s) = v_k$, then $\delta(v_j, e_{j,k}, -s) = v_i$. This implies that a sequence of moves is invertible, and can be retraced. A marker operation is fully specified by indicating whether it is being picked up, put down, or not operated upon. A *simple action* is defined as a marker operation accompanied by (followed by) a move, i.e., the robot performs some operation on the marker in the current vertex and then moves to a new location.

Perception A robot's perception is of two kinds, *marker-related* and *edge-related* perception. *Marker-related* perception enables a robot to sense whether the marker is present at the current vertex. With *edge-related* perception, a robot can determine the relative positions of edges incident on the current vertex v_i in a consistent manner, e.g., by a clockwise enumeration for a planar graph. As a result, the robot can assign an integer label to each edge incident on v_i, representing the order of that edge with respect to the edge enumeration at v_i. The label 0 is assigned arbitrarily to the edge through which the robot entered vertex v_i. Entering the same vertex from two different edges leads to two different local orderings, one of which is a shifting (circular translation) of the other. If the robot visits the same vertex twice, it must relate the two different local orderings produced and unify them into a single global ordering. Determining when the

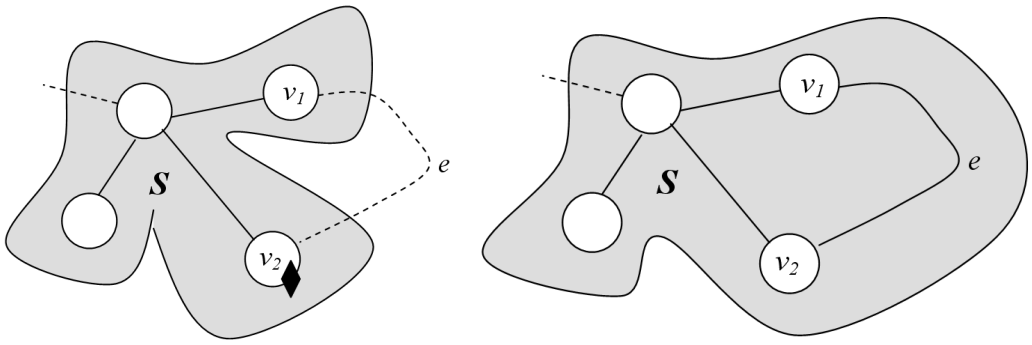
same vertex has been visited twice and generating a global ordering for each vertex is a key component of exploration.

Memory The robot remembers all raw sensory information that it has acquired and all of its actions. Specifically, if the robot has performed steps $0, 1, ..i$, the raw memory of the robot contains the sequence of information at each step. For the i-th step, it remembers marker sensing at the step, the order of edges incident on the vertex visited at step i, and the action taken at step i. By "remembering" the motion sequence, the robot may retrace any previously performed motion.

Exploration algorithm

It was shown in [14, 15] that as long as the explorer is equipped with a single unique marker that can be dropped and picked up at will it is possible for a robot to fully map its environment. The core technique is to validate (disambiguate) locations that could be confused. This is achievable due to the fact that the path taken by the robot can be retraced, and that the marker is unique.

The following description of the exploration algorithm follows that presented in [14]. The basis of the exploration algorithm is the maintenance of an explored subgraph of the full graph. As new vertices are encountered, they are added to the explored subgraph, and their outgoing edges are added to the set of edges that lead to unknown places and therefore must be explored. More formally, the algorithm maintains an explored subgraph S, and a set of unexplored edges U, which emanate from vertices of the explored subgraph. A step of the algorithm consists of selecting an unexplored edge $e = (v_1, v_2)$ from U, and "validating" the vertex v_2 at the unexplored end of the edge (the other vertex v_1 incident upon e is already in the subgraph S). Validating a vertex v_2 means making sure that it is not identical to any other vertex in the explored subgraph. This is carried out by



(a) Marker found, v_2 is known

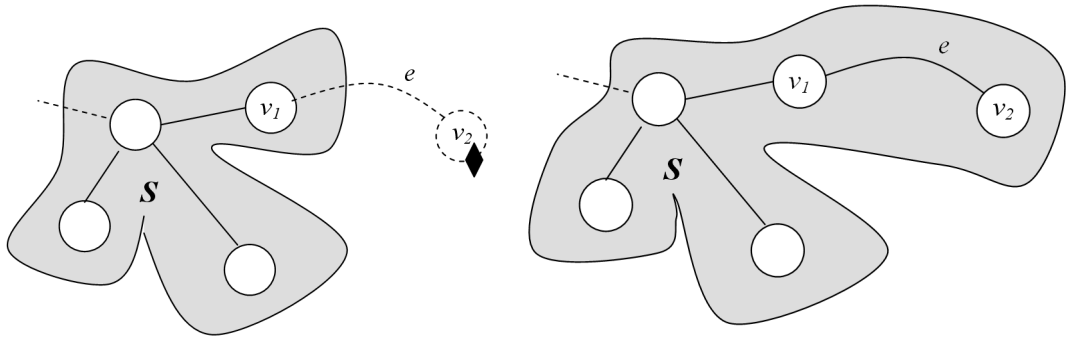
(b) S augmented by new edge e

Figure 2.4: Marker found, S augmented by adding new edge e (dotted and solid lines represent unexplored and explored portion of the world respectively. The marker is denoted by a ⧫).

placing the marker at v_2 and visiting all vertices of the known subgraph S along edges of S, looking for the marker. There are potentially two possible situations: the marker is found somewhere in S, and the marker is not found in S.

If the marker is found at vertex v_i of the explored subgraph S (Figure 2.4(a)), then vertex v_2 (where the marker was dropped) is identical to the already known v_i (where the marker was found). In this case, edge $e = (v_1, v_2)$ must be assigned an index with respect to the edge ordering of vertex v_i. To determine this, the robot drops the marker at v_1 and goes back to v_2 along the shortest path in the explored graph S. At v_2, the robot tries going out of the vertex along each of its unexplored incident edges. One of the unexplored edges will take the robot back to v_1, which the robot will immediately recognize due to the presence of the marker. Note that the index of e with respect to the edge ordering of v_1 is known by construction. Edge e is then added to the subgraph S and removed from U (Figure 2.4(b)).

If the marker is not found at one of the vertices of S (Figure 2.5(a)), then vertex v_2



(a) Marker not found, v_2 is a new vertex (b) S augmented by vertex v_2 and edge e

Figure 2.5: Marker not found, S augmented by adding new vertex v_2 and new edge e.

is not in the subgraph S, and therefore is a new vertex and must be added to S. The previously unexplored edge e is also added to S, which has now been augmented by one edge and one vertex (Figure 2.5(b)). Adding the vertex to the subgraph causes all edges incident upon it to be assigned an index with respect to the edge e by which the robot entered the vertex (edge e is assigned index 0) and the new edges are added to the set of unexplored edges U. Note that no other edge of the new vertex has been previously added to the subgraph, because otherwise v_2 would have already been in the explored subgraph. This index assignment establishes the edge ordering local to v_2. The algorithm terminates when the set of unexplored edges U is empty. A formal proof of correctness of the algorithm is presented in [14].

Performance Metric

Certain steps of the above algorithm are executed mechanically (e.g. edge traversals) while others are executed electronically as the robot reasons about the model. As the time constant associated with moving a robot is considerably larger than that associated with

a computational step, mechanical complexity is the limiting factor in the performance of the algorithm. For the above single robot and the extended multiple robot algorithm (next section) the mechanical time complexity of the task is the critical measurement of performance. Assuming one mechanical step for the traversal of one edge, the main cost of exploring the graph in terms of edges traversed by the robot (mechanical complexity) is $O(MN) \leq O(N^3)$, where N is number of vertices and M is number of edges [14]. This cost comes from the need for the robot to go back to its known sub-graph and visit all of the locations there to solve the key problem of 'have I visited here earlier?'. Note that this problem is closely related to the challenging problem of "loop closing" in the probabilistic *SLAM* literature.

2.5 Mapping and localization with multiple robots

Most robotic systems consider a single robot for the exploration and mapping task. In recent years, there has been increased research interest in systems composed of multiple robots that exhibit cooperative behavior. Exploring an unknown environment with teams of mobile robots is suggested to have several advantages over single robot system and leads to a variety of design tradeoffs (see [8, 18, 19, 44]). A team of robots may be able to accomplish more complex tasks than a single robot that works alone: By working in parallel, cooperating robots have the potential to accomplish a single task faster than single robot; A team of robots can be expected to be more fault-tolerant than only one robot. Due to the possible redundancy of the system of robots, destruction of a single member of a large teams of robots may not be catastrophic while the failure of the robot in a conventional single robot system is usually disastrous; The individual robots in multi-robot systems could have a simpler design than a larger, single robot, both in terms of the physical appearance (hardware) and computational ability (software). Therefore it may cost less to construct and maintain the multi robot system. In addition,

the constructive, synthetic approach inherent in cooperative mobile robotics may yield insights into fundamental problems in the social sciences (organization theory, economics), and life sciences (theoretical biology, animal ethology) [8].

Deployment of multiple robots also introduces a number of challenges. To make the collective design reliable, robust and efficient, fundamental issues have to be addressed including communication and coordination, synchronization, task division, motion planning, rendezvous scheduling, architecture design and task reconfiguring. Practical considerations such as limits of communication and interference of sensor readings further contribute to the complexity of a multi-robot system.

Compared with the single robot system, the field of cooperative autonomous mobile robotics is still new enough that no topic area within this field can be considered mature and supporting theory is still in its formative stage. The study of multiple robot systems naturally extends research on single robot systems, but is also a discipline unto itself [8, 1]. Some research illustrates theoretical accomplishments, while others presents practical embodiments. Most proposed algorithms so far address a specific component (problem) necessary for the multiple robot system. These include robot-robot communication (e.g., [2]), rendezvous scheduling (e.g., [50]), merging of partial maps (e.g., [33]) and task allocation during exploration (e.g., [47], [60]). In spite of the variety of the frameworks and solutions proposed in these research areas, they share the same main idea, i.e., to coordinate the work of robots to achieve better performance, e.g., improved accuracy of mapping and localization, reduced exploration time. Reviews of earlier work in multi-robot systems can be found in [8] and [19]. More recent work is reviewed in [24], [44] and [1].

Taxonomies and other theoretic work Research has been carried out in an effort to develop the classification of robot collaboration research by defining a taxonomy or

Axis	Description
Collective Size	The number of autonomous agents in the collective.
Communication Range	The maximum distance between two elements of the collective such that communication is still possible.
Communication Topology	Of the robots within the communication range, those which can be communicated with.
Communication Bandwidth	How much information elements of the collective can transmit to each other.
Collective recongurability	The rate at which the organization of the collective can be modified.
Processing Ability	The computational model utilized by individual elements of the collective.
Collective Composition	Are the elements of the collective homogeneous or heterogeneous.

Table 2.1: Robot collective taxonomy proposed in [19].

collection of research axes [8, 18, 19, 13, 24, 44]. The objective of each of these taxonomies is to provide a common language for the description of seemingly disparate theoretical and practical results, to clarify the strengths, constraints and tradeoffs of various designs, and more importantly, to highlight various design alternatives and research directions. For example, the taxonomy described in [8] identifies *group architecture*, *conflict resolution strategy*, *origins of cooperation*, *learning* and *geometric problem* as 'research axes' within which different systems can be compared. The alternative taxonomy provided in [18], [19] and [13] emphasizes design dimensions for communication and coordination, for which a finer granularity is defined in order to highlight the importance of different communication strategies on the overall capacity of the collective (see Table 2.1 for the axes and their descriptions). For example, for communication range, the authors list three key classes for the dimension. These include *COM-NONE*, i.e., robots cannot communicate with other robots directly, *COM-NEAR*, i.e., robots can only communicate with other robots which are sufficiently nearby and *COM-INF*, i.e., robots can communicate with any other robot. For communication bandwidth, sample points along this dimension include *BAND-INF*, i.e., communication bandwidth is sufficiently high that the communication cost and overhead can be ignored (communication is free), *BAND-MOTION*, i.e., communication costs are of the same order of magnitude of the cost of moving the

robot between locations, *BAND-LOW*, i.e., communication costs are very high and are much more than the cost of moving from one location to another and *BAND-ZERO*, i.e., no communication is possible. The taxonomy described in [44] instead organizes the research by the principal topic areas that have generated significant levels of study, including *biological inspirations*, *communication*, *architecture* and *localization* etc. Rather than characterizing architectures, some work seeks to categorize the underlying problems involved in multi-robot system, such as task allocation [28] and communication [2, 43]. For example, [28] provides a taxonomy on task allocation in multi-robot systems. The work proposes the following three axes for use in describing multi-robot task allocation (MRTA) problem:

- *single-task robots (ST) vs. multi-task robots (MT)* - each robot is capable of executing at most one task at a time (ST) or some robot can execute multiple tasks simultaneously (MT).

- *single-robot-tasks (SR) vs. multi-robot tasks (MR)* - each task requires exactly one robot to achieve (SR) or some task can require multiple robots (MR).

- *instantaneous assignment (IA) vs. time-extended assignment (TA)* - if the available information permits only instantaneous assignment of tasks (IA), or includes information for tasks that can be assigned in the future (TA).

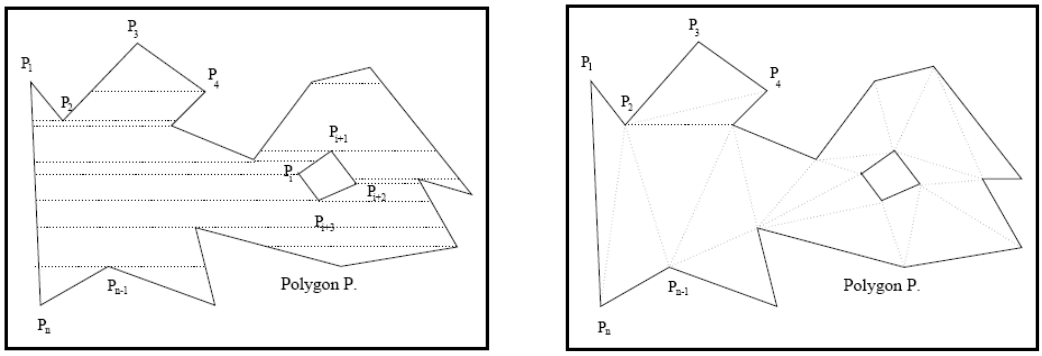
Under this taxonomy, a particular MRTA problem is defined by a combination (triple) of each of the axes. For example, a problem in which multi-robot tasks must be allocated once to single-task robots is designated *ST-MR-IA*. There are eight problems (combinations) allowed by these axes. For each problem, relevant theories and heuristic solutions are discussed. The effects of different kinds of communication on the performance of a multi-robot team in a variety of tasks is analyzed in [2]. The work distinguished three different types of communication, i.e., *No Communication*, i.e., no direct communication

among robots, *State Communication*, i.e., robots are able to detect the internal state of other robots, and *Goal Communication*, which involves the transmission and reception of specific goal-oriented information. The work examines the extent to which various amounts of shared information facilitate certain simple multi-robot tasks, and how task and environment can affect communication payoffs. For example, if robots are grazing (consuming) some widely distributed resource to what extent is it helpful to have them explicitly transmit information on which regions have already grazed? While the work has concluded that communication provides certain benefits for particular types of tasks, it also suggested that the extent to which it does so (or sometimes fails to do so) must be carefully weighted against the additional cost of transmitting the information.

2.5.1 Coordination among robots through task allocation

While some research addresses issues such as rendezvous scheduling and merging partial representations (discussed next), a considerable amount of work addresses strategic task allocation to achieve coordination of robots during exploration. Broadly speaking, enforcing explicit coordination and cooperation among robots during exploration serves two purposes: improving the accuracy of mapping and localization, and reducing the exploration time.

Improving mapping and localization accuracy Several researchers have studied the problem of using multiple robots to reduce the localization and mapping error during exploration (e.g., [17, 47, 46, 49]). For example, the work described in [47], [46] and [49] focuses on the problem of reducing the odometry error during exploration. In [47], the world is modeled as a set of simple polygonal boundaries. Exploration is performed by two robots that jointly assist one another. Each robot is equipped with two sensors. The first sensor is an *object detector*, able to detect any object in the immediate vicinity of the



(a) Trapezoidation of a simple polygon with holes

(b) Triangulation of the same polygon

Figure 2.6: Decomposition of polygons. Courtesy of I. Rekleitis [47].

robot. The second sensor is a *robot tracker* with the ability to locate another robot when there is a free line of sight between them, and to report accurately the distance to the second robot and its orientation. The robots explore the unknown environment by progressively covering free space in the polygonal world. Systematic exploration is achieved using planar decomposition [42, 45], i.e., trapezoidation and triangulation (Figure 2.6). In the work, trapezoid decomposition is used for large areas ensuring an exploration strategy that finishes with the total free space mapped as a set of trapezoids. For small areas, triangulation of the free space is used. Reduction in the size of odometry errors is accomplished by having only one robot move at any time, while the other robot(s) observes it. The stationary robot acts as an artificial landmark in order for the moving robot to recover its pose with respect to it. The procedure is referred to as *cooperative localization*. Moreover, when one robot is moving and maintains an uninterrupted line of visual contact with the stationary robot, it effectively maps the area covered by the line of visual contact. Later on the roles are reversed: the robot that had been moving becomes

the observer while the other robot moves. The algorithm assumes full communication, which enables the moving robot to obtain its current position from the observer(s) at any time. Cooperative localization is also used in [49] where cooperative localization is used to explore the visual domain.

Reducing exploration time While the above approach can reduce the odometry error, it is not designed to distribute the robots over the environment, so exploration time may not be reduced. There has been considerable work done in addressing explicit coordination during exploration to achieve reduced exploration time. The basic idea behind the work is to allocate the task in such a way that redundant work is avoided or reduced (see [60, 6, 51]). Consider the work of [60, 51]. The key idea here is to explicitly coordinate the robots so that they simultaneously explore different regions of their environment. In [60] a technique is described in which robots build a common map in a distributed fashion. The work introduces the notion of a *frontier*. Frontiers are regions on the boundary between open space and unexplored space. An occupancy grid is used as the spatial representation. Each cell of the grid stores the probability that the corresponding region in space is occupied. A process analogous to edge detection and region extraction in computer vision is used to find boundaries between open space and unknown space. Any open cell adjacent to an unknown cell is labeled a frontier cell. Adjacent edge cells are grouped into frontier regions. Any frontier region above a certain minimum size (roughly the size of the robot) is considered a frontier. Once frontiers have been detected within a particular grid, the robot attempts to navigate to the nearest accessible, unvisited frontier using a depth-first search on the occupancy grid. Each robot has its own global occupancy grid that represents its own knowledge about the environment. Whenever a robot arrives at a new frontier, it sweep its sensor and constructs a local occupancy grid representing its current surroundings. The novelty of the work is that the robot also broadcasts the local

grid to all of the other robots. Each robot stores the local grid received from other robots. When a robot arrives at a new froniter, it integrates these received local grids with its global grid along with the new local grids it constructs (by sweeping its sensor) at the frontier. This has the advantage of being both cooperative and decentralized. All of the information obtained by any robot is available to each robot. This allows robots to use the data from other robots to determine where to navigate. Based on this information, a robot can determine which areas have already been explored by other robots and then choose to explore an unexplored region. A robot can also discover that a frontier detected by another robot is nearby and decide to investigate. While information is shared, control is independent, allowing the team to be robust to failures of individual robots. This approach, however, has its limitations. Since navigation is independent, robots may waste time by navigating to the same frontier, i.e., there is no coordination component which chooses different frontiers for the individual robots. This work is extended in [6] and [51] by coordinating the robots so that they don't choose the same frontier, significantly decreasing the time needed to accomplish the exploration task. For example, the work in [51] considers two coordination problems: creating a single global map from the sensor information of the individual robots (*coordinated mapping*), and deciding where each robot should go in order to create the map most effectively (*coordinated exploration*). The basic approach to both coordination problems is similar: distribute most of the computation amongst the individual robots and integrate their results asynchronously by using centralized modules performing efficient global computations over the data. This operates under the assumption that the robots know their pose relative to one another and have access to high-bandwidth communication. These two coordination problems are interrelated during the exploration process, as we show below. For coordinated mapping, each robot processes its own laser data to create a consistent local map. The laser data is communicated to the central mapper. The central mapper module then integrates the

local maps received to create a consistent global map, which is sent to the individual robots. The central mapper improves the map (minimizes localization error) by iteratively combining data from the robots. Similarly, the approach to coordinated exploration combines distributed computation with global decision making. The individual robots construct "bids", which describe their estimates of the expected information gain and costs of traveling to various frontier cells from the current position. The bids are sent to a central executive, which assigns tasks to each robot based on all the bids received. The cost in the bid is computed based on the optimal path of visiting a frontier cell from the current position. The information gain is evaluated based on the number of unknown cells from the frontier cell that the robot can sense. A robot constructs a new bid each time it receives a map update from the central mapper. The central executive receives the bids and assigns tasks based on all the bids, attempting to maximize overall utility, while trying to minimize overlap in coverage by the robots. Thus while a robot may prefer to visit one location the executive might assign it a different location if another robot is expected to gain much the same information.

2.5.2 Rendezvous problem

There is also research that addresses the problem of schedulling the rendezvous of individual robots. [50] considers the problem of rendezvous between two robots exploring an unknown environment. That is, how can two autonomous exploring robots that cannot communicate with one another over long distances (i.e., COM-NEAR) meet if they start exploring at different locations in an unknown environment? This work is the first to formalize the characteristics of the rendezvous problem, with limited communication and unknown starting places. In the simplest, idealized noise free case, if the robots have a pre-arranged notion of what constitutes a good rendezvous point, the rendezvous problem would be simple, as the following steps show.

1. Travel throughout environment.

2. Find good rendezvous locations (landmarks).

3. At the pre-arranged meeting time, choose the best rendezvous location.

4. Travel to that rendezvous location, and share information with the other agents.

In practice the problem with this idealized scenario is that due to sensor variations, or disjoint landmark sets, the robots may not agree on where the ideal landmark is situated, or may not arrive at the landmark as scheduled (at the pre-arranged meeting time). The work examines how to choose good rendezvous locations, what can cause a rendezvous attempt to fail, and how to recover from failure. The work first models the environment as a function of the sensors. This function gives rise to a distinctiveness surface, defined over the domain of the environment. The work then chooses landmarks at the local extrema of the surface, limiting the knowledge of the surface only to those points that the agents have visited. Which points the robot visited is dictated by the trajectory prescribed by the underlying task. The robot assigns a value to every point and orders the resultant landmarks in the environment in terms of distinctiveness. This ordering allows the landmarks to be ranked in terms of their likelihood to lead to a successful rendezvous. To develop robust rendezvous strategies that permit a robot to interleave exploration and attempted rendezvous so that even if the rendezvous fails, the robots can continue exploring, the authors identify the following attributes that characterize the rendezvous problem:

- *Similarities (sensor noise)* - the reproducibility of the perceptions between robots. Due to sensor noise, distinctiveness measures observed by the two robots are unlikely to be identical. This leads to strategies that must consider a larger number of candidate rendezvous landmarks since a single guaranteed candidate may not be determined reliably.

- *Landmark Commonality* - the extent of overlap between the spatial domains of the agents. The robots may have explored different areas and may have selected different landmarks that are not in the common region, i.e., the landmark sets are not identical. The effect of the non-commonality is that both robots must consider a larger subset of candidate landmarks, since any given landmarks selected by one robot may not be known to the other robot.

- *Synchronization* - the level of synchronization between the agents. If the agents do not agree on the rendezvous time, or have delays in travel to the rendezvous place, there is the possibility that the rendezvous will be missed. This is called *asynchrony*. This effect leads to a need for strategies that may revisit the same landmarks repeatedly to compensate for missed meetings.

Two main classes of algorithms (deterministic and probabilistic) are described in [50] along with examples of each class of algorithm. The deterministic class of algorithm creates a list of all possible combination of landmarks and specifies the order in which the landmarks should be visited. There is no random aspect to the landmark visit sequence, and therefore the algorithms generate the same sequence of landmark visits for a given landmark set. The probabilistic class of algorithm does not generate an *a priori* ordering of landmarks but rather simply generates probabilities for landmarks being visited at any proposed rendezvous. For example, the simplest deterministic sequential algorithm would be: 'one agent picks a landmark and waits there for the other agent, which visits every landmark in turn. If the second agent has visited every landmark without encountering the first agent, the first agent moves to another landmark it has not yet visited.' In order to determine the characteristics of the algorithms, the work provides a closed-form analysis of the worst and expected-case complexity of the algorithms at points in the attribute (parameter) space. The closed-form analysis is complemented by a numerical description of the performance of the algorithms at a range of points in the parameter space. An

interesting conclusion from these results is that depending on a combination of these attributes (confounding factors) no strategy is canonically a good or bad choice. Each of the algorithms has particular advantages and disadvantages. Different algorithms have their own domains of superiority. For example, the deterministic sequential algorithm is simple and is preferable when asynchrony is low. The probability approach, on the other hand, outperforms the other algorithms when asynchrony is a problem but sensor noise is not.

2.5.3 Merging partial world representations

An algorithm for merging two topological maps obtained by robots having *no* common reference frame or global positioning, assuming rendezvous has occurred is represented in [33]. The spatial representation mechanism is similar to that modeled in [14, 15], i.e., representing the environment as embedded topological maps where vertices represent "places" and edges represent paths between the places. Edges incident to a vertex in a topological map have spatial interrelationships, e.g., clockwise ordering for planar case. In contrast with Dudek et al.'s model in [14, 15], the robot in the work can measure orientation and the distance between places. So edges and vertices in maps have attributes such as length and global orientation. The algorithm proceeds in two phases and is inspired by methods from graph matching [5] and image registration [26]. The first phase of the algorithm identifies hypothesized matches, i.e., possible sets of vertex and edge pairs, between the two maps. These correspond to common connected subgraphs of the maps and reflect areas of the environment with identical structure. In testing the compatibility of vertex and edge pairs, exactly known attributes of paired vertices or edges such as the degree of vertices in a static world, should match perfectly. Attributes that are subject to measurement error, such as the length of an edge or the angles between edges leaving a vertex are compared using a similarity test that takes into account the

relevant error model. The second phase of the algorithm considers the global geometry of the hypothesized matches. For each hypothesis, the algorithm estimates the geometric transformation that must be applied to one map in order to bring paired vertices into alignment. Geometrically compatible hypotheses give rise to similar transforms, so the hypotheses can be clustered in the transformation space. The best cluster (based on size and error) is returned as the algorithm's result, and the vertex and edge correspondences from that cluster are used to merge the maps. The algorithm has been demonstrated on simulated and real-world maps. Results show that the algorithm is most sensitive to discrepancies between the error model used and the true error in the maps. The algorithm performs well when there was less error than assumed. When there is more error in the map than assumed, the algorithm performs poorly. According to the authors, the reason of the poor performance is that the correct hypothesis may be rejected. This is because under the underestimated error model, corresponding edges or vertices of some correct hypotheses appear too different so they are treated as incorrect hypotheses and therefore are mistakenly rejected.

2.5.4 Extending SLAM to multiple robot systems

Although not closely related to the work in this report, it is worth mentioning that there exist a number of multiple robot SLAM formulations based on the Kalman filter and its derivatives [25, 58, 59] and on the particle filter and its derivatives [54, 32]. [25] takes the "classical" approach of generalizing the extended Kalman filter (EKF) SLAM formulation to handle multiple robots. Similarly, [58] and [59] reformulate the SLAM problem using the sparse extended information filter, which has some inherent advantages over EKF. The information filter is then generalized for the multi-robot case. The work in [54] describes a hybrid algorithm that combines maximum likelihood mapping with particle filters. To address the challenging problem of determining the initial poses of the robots, the work in

[32] (which in part is a generalization of the work in [54]) proposes an online algorithm for multiple robot SLAM. The starting point is the single-robot Rao-Blackwellized particle filter [31]. The particle filter is extended to handle multi-robot SLAM problems in which the initial pose of the robots is known, and an approximation is introduced to solve the more general problem in which the initial pose of robots is not known *a priori* (such as occurs when the robots start from widely separated locations).

2.5.5 The work of Dudek et al.

Finally we present the work by Dudek et al. on multiple robots [20], which is the starting point for the work in this report. The work in [20] presents an informal extension of the algorithm in [14, 15] to the case of multiple robots. This extension assumes the same environmental representation as described in [14, 15] and populates the world with two or more robots, each of which is equipped with its own marker. Due to the deployment of multiple robots, some definitions and assumptions are different from those in [14, 15] and these are described below.

The Model

The World The same environmental representation as [14] is assumed, i.e., the environment is represented as an embedding of an undirected graph, that is, as a fixed set of discrete locations or regions with an ordered set of bidirectional paths between them. In addition, it is assumed that each node has sufficient space to hold multiple robots (so robots can meet in a node) and multiple markers (so different markers can be dropped in the same node).

Perception A robot's perception is of three kinds: *edge-related*, *marker-related*, and *robot-related*. The *edge-related* perception is same as described in [14], i.e., a robot can

determine the relative positions of edges incident on the current vertex v_i in a consistent manner. As a result, the robot can assign an integer label to each edge incident on v_i, representing the order (relative position) of that edge with respect to the edge enumeration at v_i. The marker-related perception of a robot allows a robot to sense whether or not its unique marker is present at the current vertex. It also allows it to sense whether or not other robots' markers are at the current vertex. With robot-related perception, a robot is able to sense the presence of the other robot(s) at the current vertex (robots can meet in a vertex and sense each other when they meet).

Movement and marker operation A robot can move and handle its own marker. As in [14], a robot moves from one vertex to another by traversing an edge (a *move*). Note that to ensure the fairness of robots, one robot cannot pass another in an edge. Again, it is assumed that a sequence of moves is invertible and thus can be retraced. A robot can put down the marker it holds at the current vertex and it can pick up its marker that is located at the current vertex (a *marker operation*). Note that this implies that a robot can sense the others' markers but only manipulates its own marker.

Robot communication A robot can communicate with other robot(s) when they are at the same vertex. It is assumed that all robots in the same vertex can communicate with each other, and that there exists sufficient bandwidth and synchronization strategies for this to take place. Robots may only communicate when they are in the same vertex.

We can see that according to the axes in the taxonomy proposed in [19], the model can be specified by COM-NEAR (robots only communicate when they are sufficiently nearby), BAND-INF (communication is free), TOP-ADD/BROAD (robots communicate by broadcast, or by robot name and address). In addition, in the model, robots are homogeneous (COMP-HOM), the relationship between members of the collective can change arbitrarily (ARR-DYN). Compared with the model in [33], this model does not

allow measurement of orientation and edge length, so there are no edge lengths and angles in the model. On the other hand, since the robots start at the same location, robots have a common reference frame.

Outline of the exploration algorithm

The following description of the exploration algorithm follows that presented in [20]. Joint exploration is achieved through alternating phases of independent exploration and coordinated merging of the independently acquired sub-graphs. At any time, the robots retain a common representation of some part of the world (the commonly known subgraph) as well as additional information regarding other parts of the world that the other robot may not be aware of. The algorithm proceeds by having the robots start at a single location with a common reference direction (the initial definition of the common representation), and partitioning the unknown edges leaving the commonly known representation so that each robot explores independently, using the exploration algorithm sketched in [14, 15]. After exploring for a previously agreed-upon interval, the robots return to a commonly known location to merge their individual partial world models, after which the merged map is shared between the robots, becoming the new commonly known representation and the remaining unknown portions of the common map are re-partitioned between the robots and the entire algorithm repeats until the environment is fully explored.

The key step in the multiple robot exploration algorithm is the merge phase. Similar to the strategy for single robot exploration [14, 15], the merging of partial maps is accomplished by visiting portions of one map and disambiguating them with portions in the other map. The main techniques in [20] including the merging technique will be discussed in Chapter 3, which builds on and extends the work in [20].

2.6 Open problems

Dudek et al.'s single robot exploration algorithm [14, 15] and multi-robot algorithm [20] suggest several interesting extensions and variations. Most of these variations and problems are related to the core techniques involved in single and multiple robot exploration, including details related to the search for disambiguation, merging the partial representations obtained by multiple robots, partitioning of the world to be explored, and rendezvous scheduling.

Search Strategy The main cost of exploration is the cost associated with the search for the marker used for disambiguation. In [14, 15] and [20] a brute force strategy is adopted, i.e., each incoming search task is performed, irrespective of how 'hard' it is. In a connected bidirectional graph a new node or edge can be approached and validated via different routes with different costs. It is worthwhile exploring more sophisticated search techniques. For example, a 'lazy exploration' approach, in which 'hard' search tasks are put off to later phases may produce significant improvement to the cost of exploration. How to predict the difficulty of a specific search task and to what extent (how far) to put off the task are interesting and challenging problems. Another approach is a 'breadth-first' exploration, in which unknown places that are immediate neighbors of a new place are fully explored before more remote unknown neighbors. How to adaptively adopt this approach and the current depth-first exploration so as to minimize the search cost is an interesting problem worth investigating.

Disambiguation Strategy A problem closely related to search cost is the electronic disambiguation process before the search. The algorithm in [14] and [15] use the vertex degree to disambiguate vertices before doing mechanical search. Vertices that don't have the same degree (signature) as the marker-dropped place are identified as not being

potentially confusing and are not visited. All vertices having the same degree as the marker-dropped place (and having unexplored exit(s)) are identified as being potentially confusing and are searched. If we can, before the search, 'electrically' disambiguate more potentially confusing vertices, then the subsequent search can be alleviated or even avoided. The more information we have to describe a vertex, the more likely it is that we can disambiguate vertices. One piece of information that can be exploited is local neighbor information. This strategy is especially appealing in the merge phase, where portions of the world being disambiguated are known (explored by different robots) and therefore significant neighbor information is available.

Rendezvous Scheduling As one of the core techniques in multiple robot exploration, rendezvous scheduling defines how, when, and where the merge should be performed. The algorithm in [20] adopts a very simple schedule, e.g., to merge after an arbitrarily chosen fixed interval, and at the initial starting place. It is worthwhile investigating issues related to rendezvous scheduling, e.g., how different rendezvous intervals and locations can affect the performance, and how these factors are related to the underlying graph and the current partial maps, how to adaptively decide subsequent intervals and locations based on the results from previous intervals.

Merge Strategy In the merge phase of multiple robot exploration, coordination and parallelism should be explored. In [20] the merge task was performed by one robot. To achieve higher utilization of robots and lower execution cost, it is worthwhile parallelizing the merge task by having other robots involved, e.g., to share the search and validation task.

How to merge subgraphs when more than two robots are deployed is another problem, especially when a large number of robots are deployed. Parallelism and coordination should be explored. For example, the algorithm described in [20] adopts a simple sequen-

tial merge, in which the order is chosen randomly. Approaches such as a binary partitioned merge might be explored. In addition to exploiting parallelism, more strategic techniques should be explored to exploit the availability of multiple robots, e.g., dispatching some robots to the known graph to facilitate search and validation.

Evaluation mechanism Finally we note that an evaluation mechanism needs to be defined for the general multiple robot exploration problem. An evaluation mechanism for the multiple robot exploration algorithm is not as simple as that in the single robot case. For example, suppose we wish to evaluate the mechanical cost of performing the exploration task. For the single robot case (e.g., [14, 15]), the cost of the task is the mechanical cost of the robot. In the multiple robot case, there are both parallel and sequential work involved, due to possible synchronization mechanisms used. For example, when multiple robots are involved in the merge task, they may need to meet during the parallel work, then start parallel work again. We need to define a general evaluation framework to accommodate for possible interleaving of parallel and sequential work.

2.7 Summary

This chapter defined formally the problem to be addressed, i.e., robotic exploration and mapping problem in a graph-like world where multiple robots are deployed. It reviewed approaches in the literature to related problems, beginning with an introduction to the problem. Topics discussed include the interrelated problem in robot mapping (mapping and localization), basic spatial representations (metric and topological representation) and the pros and cons of each of the representations, and deterministic and probabilistic representation schemes. Metric approaches to the mapping problem were presented with an emphasis on approaches to solving the mapping and localization problems simultaneously within a probabilistic framework. The posterior of the SLAM problem was

presented and two approaches that estimate the posterior, i.e., Kalman filter and particle filter were discussed. Topological approaches to the problem were reviewed with an emphasis on Dudek et al.'s deterministic topological approach that forms the basis of the algorithms developed in this report. The model, algorithm details and performance concerns of Dudek et al.'s approach were reviewed. The chapter also presented a survey of multiple robot exploration systems, including Dudek et al.'s multiple robot exploration algorithm, upon which the work in this report is built. The chapter concluded with an overview of some of the open problems in single and multiple robot exploration using a topological representation. This includes how to measure the performance of multiple robot system, how to perform rendezvous scheduling, how to perform exploration in a strategic way, how to split up tasks, and how to perform merging when large groups of robots are involved. The next chapter formally extends Dudek et al.'s model and their approach to the case of multiple robots.

3 Multiple Robot Exploration

This chapter begins by defining formally the models used in this work and then provides a detailed description of the multiple robot exploration algorithm, with particular emphasis on the process of merging the partial representations obtained by individual robots. The chapter also presents a detailed formalism of one possible merge algorithm, followed by a proof of its correctness and sample operations. In order to compare different solutions, a performance metric for multiple robot exploration is developed as well.

3.1 The model

The task addressed in this report is that of mapping in a graph-like world using a collection of robots with limited communication abilities, with all of the robots initially deployed in a common location. We begin by developing a formal definition of the world in which the robots exist and which is to be mapped.

3.1.1 The world

The same environmental representation as [14] is assumed. For completeness of definition, we briefly present it here. The environment is represented as an embedding of an undirected graph $G = (V, E)$ with set of vertices $V = \{v_1, ..., v_n\}$ and set of edges $E = \{(v_i, v_j)\}$. The world is assumed to have no multiple edges between two vertices and no edge incident twice at the same vertex. The definition of an edge is extended to

allow for the explicit specification of the order of edges incident upon each vertex of the graph embedding. This ordering is obtained by enumerating the edges in a systematic (e.g. clockwise) manner from some starting direction. An edge $e_{i,j}$ incident upon v_i and v_j is assigned labels n and m, one for each of v_i and v_j respectively. n and m represent the ordering of the edge $e_{i,j}$ with respect to the consistent enumeration of edges at v_i and v_j respectively. The labels n and m can be considered as local directions, e.g., from vertex v_i then n'th exit takes edge $e_{i,j}$ to vertex v_j. A route (path) can be specified as a sequence of edge labels such that the entry edge at a vertex is always the reference edge and the successive labels specify the exit edges. Our model assumes that each vertex has sufficient space to hold multiple robots (so that robots can meet in a vertex) and multiple markers (so that different markers can be dropped in the same vertex).

3.1.2 Perception

A robot's perception is of three kinds: *edge-related*, *marker-related*, and *robot-related*.

Edge-related perception The *edge-related* perception is the same as described in [14]. Assume that the robot is at vertex v_i (having arrived via edge $e_{j,i}$), the robot can determine the relative positions of edges incident on the current vertex v_i in a consistent manner, e.g., by a clockwise enumeration starting with $e_{j,i}$. As a result, the robot can assign an integer label to each edge incident on v_i, representing the order of that edge with respect to the edge enumeration at v_i. The label 0 is assigned arbitrarily to the edge $e_{j,i}$, through which the robot entered vertex v_i. Thus for each vertex v_i the robot defines a mapping m that maps v_i's incident edge $e_{i,k}$ onto an integer $m(e_{i,k})$, where $0 \leq m(e_{i,k}) < degree(v_i)$. This mapping depends on the entry edge $e_{j,i}$. Entering the same vertex from two different edges leads to two mappings one of which is a permutation of the other. As a byproduct of this ability, the robot can also sense the degree of the

current vertex. We also assume that all robots follow the same consistent edge enumeration rule. This ensures the comparability and computability of edge ordering of different vertices in the map.

Marker-related perception The *marker-related* perception of a robot allows a robot to sense the set of markers present at the current vertex. This is the same marker-related perception as in [20]. Assume that the robot is at vertex v_i, also assume that there are K robots involved. The marker-related perception of the robot at v_i is a K-tuple $MS_i = (ms_1, ms_2, ..., ms_K)$, where each element ms_k has a value from the set {*present*, *not present*}, according whether robot k's marker is present at vertex v_i.

Robot-related perception A robot is able to sense the presence of the other robot(s) at the current vertex. Again, assume that the robot is at vertex v_i, also assume there are K robots involved. The robot-related perception of the robot is a K-tuple $RS_i = (rs_1, rs_2, ..., rs_K)$, where each element rs_k has a value from the set {*present*, *not present*}, according to whether robot k is currently present at vertex v_i.

3.1.3 Robot communication

Communication between robots is only possible when they are at the same vertex (COM-NEAR). It is assumed that all robots in the same vertex can communicate with each other (TOP-ADD/BROAD), and that there exists sufficient bandwidth and synchronization strategies for this to take place (BAND-INF). Robots may only communicate when they are present at the same vertex. A robot can only receive communication messages from robots in the same vertex.

3.1.4 Movement and marker operation

As in [14] a robot can move from one vertex to another by traversing an edge (a *move*). Assume the robot is at a single vertex, v_i, having entered the vertex through edge $e_{l,i}$. In a single move, it leaves vertex v_i for v_j by traversing the edge $e_{i,j}$, which is located r edges after $e_{l,i}$ according to the edge order at vertex v_i. This can be formally expressed by the transition function

$$\delta(v_i, e_{l,i}, r) = v_j.$$

A single move is thus specified by the order r of the edge, along which the robot exits the current vertex, where r is defined with respect to the edge, along which the robot entered such vertex. As in [14], it is assumed that a sequence of moves is invertible and thus can be retraced.

As in [14, 15], a robot can put down the marker it holds at the current vertex and it can pick up its marker that is located at the current vertex (a *marker operation*). This is specified by M_{op}, where M_{op} has a value from the set {*pickup*, *putdown*, *null*}. Note that a robot can sense the other's markers but only manipulates its own marker.

3.1.5 Memory

The robot remembers all raw sensory information that it has acquired so far, all communications that it has engaged with, and all of its actions. Specifically, if the robot has performed steps $0, 1, ..i$, the raw memory of the robot contains the sequence of information at each step. For the *i-th* step, it remembers marker sensing at the step, the order of edges incident on the vertex visited at step i, and the action taken at step i. By "remembering" the motion sequence, the robot may retrace any previously performed motion.

3.1.6 Atomic actions and synchronization

In extending the problem of graph exploration to multiple robots, a critical issue is the nature of parallelism and synchronization. Here we make the synchronization assumption that all of the robots operate in parallel but that each robot executes a specific tuple of operations as an atomic unit and that during that operation no other robot is operating. Each robot executes the following sequence

$$(sense, initiate\text{-}communications, manipulate, move)$$

as an atomic operation. That is, even though all robots operate in parallel, the sequence (sense, initiate-communications, manipulate, move) is executed as a single operation.

- *sense* – this includes marker, robot and edge-related perception.
- *initiate-communications* – the robots currently in a specific vertex may communicate. This communication is only initiated by the robot that is currently executing.
- *manipulate* – the robot manipulates its marker (pick up or put down or null).
- *move* – the robot may choose to exit the current vertex and move along an edge to the next vertex.

There are a number of properties of this model that are worth observing:

1. As a move is a complete transit of an edge, robots never meet in hallways, only at vertices.
2. Robots never manipulate each other's markers.
3. If the active robot chooses not to initiate communication then the other robot(s) in the same vertex may not have an opportunity to communicate with it.

4. We do not assume that robots get to see which edge another robot entered the vertex from. Although this would be a powerful source of information and an interesting direction for future work.

The parallelism does not assume a global clock, or that distance or velocity information is available. Thus the only 'clock' that robots have access to is the number of edges that they themselves have traversed. It is not assumed that the parallelism is 'fair' [7, 9] in that one robot may move over and over again while another robot 'sleeps'.

The assumption of an atomic sequence of action simplifies synchronization in vertices. There is no possibility of having multiple robots acting on the set of markers and 'running into' each other, or worrying about synchronization between perception and action. In practice, some sort of synchronization mechanism would have to be developed in order to permit this type of atomic operation (e.g. the use of semaphores [7, 41] or mutexes [41]). But this is beyond the scope of this work.

3.2 The multiple robot exploration algorithm

Given the set of assumptions given above, this section presents the multiple robot exploration. Fundamental issues and techniques are discussed, followed by formal description of the merge algorithm and a correctness proof and sample operations of the algorithm.

3.2.1 Joint exploration

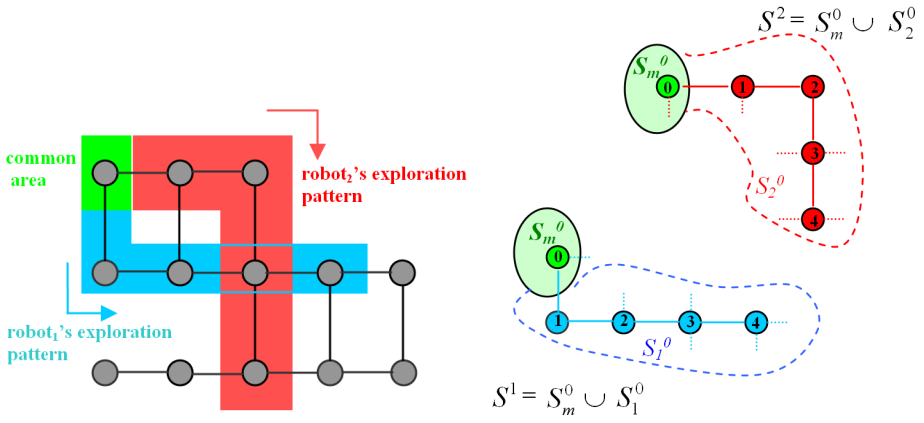
We begin by considering the problem of two robots exploring the world in concert. Extensions to $k \geq 2$ robots are examined later. Joint exploration is achieved through alternating phases of independent exploration and coordinated merging of the independently acquired sub-graphs. The algorithm proceeds by having all of the robots start at a single location with a common reference direction, and partitioning the unknown

edges leaving the known world so that each robot explores independently, using the exploration algorithm described in [14, 15]. After exploring for a previously agreed-upon interval, which is defined in terms of the number of edges traversed, the robots return to a commonly known location to merge their individual partial world models, after which the remaining unknown portions are re-partitioned between the robots and the entire algorithm repeats until the environment is fully explored. At any time during the exploration, if a robot discovers that no further unexplored edges exist in its assigned set, it heads to the rendezvous location and waits there.

3.2.2 Merging the partial representations

The challenging task of multiple robot exploration is the task of efficiently merging the partially explored environments obtained by the two robots. Each merge process takes two partial maps and involves disambiguating possible confusions between the two maps obtained by the robots. One of the partial maps is chosen as the *base map* that will be augmented with information collected by the other robot. After merging the base map with the other partial map, the augmented base map is shared among (copied to) both robots, becoming the new commonly known world representation.

Partial representations before merge process After the two robots have explored independently, robot_1's partial map S^1 consists of a commonly known part S_m^0 and the portion of the world S_1^0 that robot_1 has independently explored, i.e., $S^1 = S_1^0 \cup S_m^0$ (the superscript on S_1 and S_m signifies that this is the map before the merge process starts, i.e., at merge timestep 0). Similarly robot_2's partial map S^2 consists of the commonly known part S_m^0 and the independently explored part S_2^0, i.e., $S^2 = S_2^0 \cup S_m^0$. The problem becomes that of integrating S_1^0, S_2^0 and S_m^0. Figure 3.1(a) shows sample exploration patterns of two robots on a lattice-like world and Figure 3.1(b) shows the resulting two



(a) Exploration patterns by robots

(b) Partial maps obtained by the robots (before the merge)

Figure 3.1: Partial maps to be merged and exploration patterns generating the maps.

partial maps obtained by the robots. In the figure, explored edges are represented by solid lines connecting their end vertices, unexplored edges are represented by short dotted lines leaving their known ends. Vertices and edges in the independently explored map S_1^0 (by robot$_1$) are colored blue, edges and vertices in the independently explored partial map S_2^0 (by robot$_2$) are colored red, the common map S_m^0 is colored green. Several properties of the partial maps are worth observing. Firstly, we can see that each independent partial map has its own local specification (vertex labelling and edge ordering). In this example, both S_1^0 and S_2^0 have a 'vertex 2' but they refer to different places in the lattice world they explore. Secondly, the independent partial maps S_1^0 and S_2^0 may intersect. Here 'vertex 3' in S_1^0 and 'vertex 3' in S_2^0 refer to the same place in the real world. Determining which vertices refer to the same real world location and which vertices do not is a critical merge process task. Thirdly, both S_1^0 and S_2^0 are 'distinct' from S_m^0. This is because robot$_1$ has already disambiguated vertices and edges in S_1^0 with that in S_m^0 and robot$_2$ has already disambiguated vertices and edges in S_2^0 with that in S_m^0.

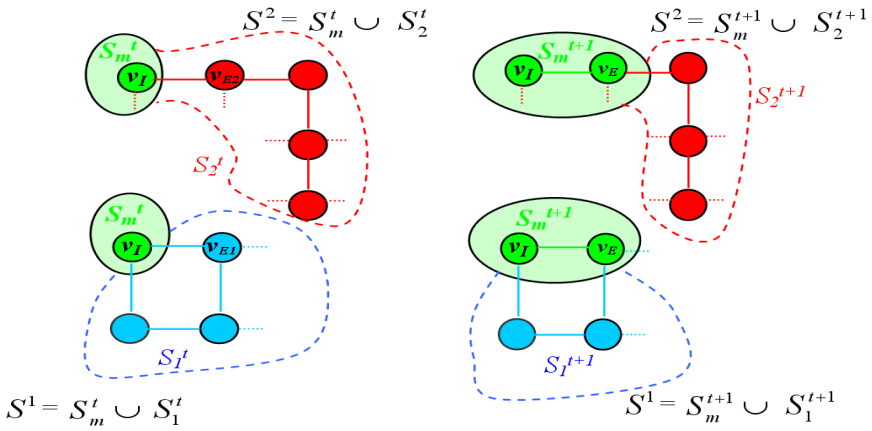
Partial presentations during merge process Clearly either S^1 or S^2 can be treated independently as being correct, so without loss of generality we choose robot$_1$'s map S^1 as the base map which will be augmented during the process. (In practice one would choose the larger map as the base map.)

Throughout the process, due to the properties of the partial maps, the merging of S^2 with S^1 involves disambiguating ('fusing') vertices and edges in S_2^t with that in S_1^t and including them into S_m^t, where S_m^t is the set of vertices and edges in the partial maps that have been merged (and therefore are commonly known) at timestep t. Here S_2^t is the independently explored part of the world in robot$_2$'s map that has not been disambiguated and included into S_m^t, and S_1^t is the independently explored part of the world in robot$_1$'s map that has not been disambiguated and included into S_m^t. So during the process, S_m^t grows and S_2^t shrinks. Upon termination (at timestep n),

- $S_1^n \cup S_m^n$ (augmented S^1) is the merged result.
- $S_2^n = \{\}$, i.e., all vertices and edges in S_2^0 have been disambiguated and included into S_m^n.
- S_1^n is the set of edges and vertices independently explored by robot$_1$ but **not** by robot$_2$ (so they are not merged into S_m^n).
- $S_m^n = S_m^0 \cup X$, where X is the set of edges and vertices independently explored by robot$_2$ (may or may not be explored by robot$_1$ as well).

In the following discussions, the superscript t will be omitted unless it is necessary.

Frontier edges and fusing of frontier edges To visit vertices and edges in S_2 in a systematic and efficient way, the algorithm starts from S_m, studying independently explored edges (by robot$_2$) that are grounded in S_m but which extend into S_2. We call such edges *frontier edges*. Formally, a frontier edge is an edge $e = (v_1, v_2)$ such that



(a) The frontier edge was also explored by robot$_1$

(b) Expanding edge $(v_I,\ v_{E2/E1})$ into S_m

Figure 3.2: Electronic fusing.

1. $e \in S_2$, i.e., e has not been merged.

2. exactly one of $\{v_1,\ v_2\} \in S_m$ (call this the interior vertex) and the other $\in S_2$ (call this the exterior vertex).

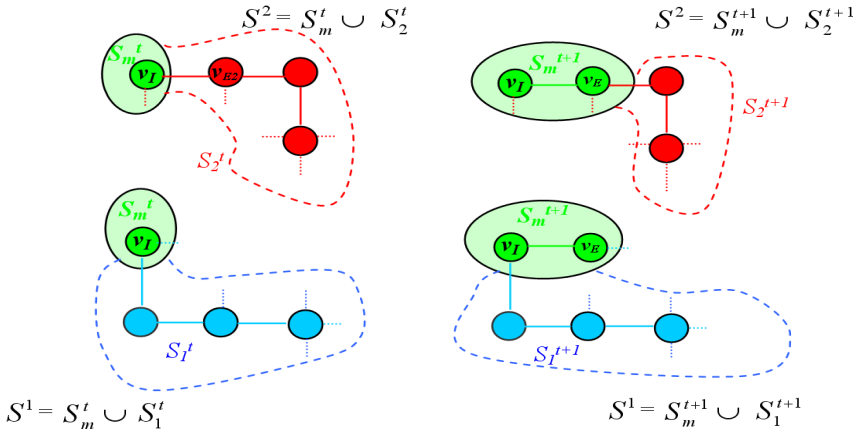
Each iteration of the algorithm involves choosing a frontier edge from S_2 and processing it. As each frontier edge is processed it is removed from the set of frontier edges (by expanding the common area S_m) and additional edges in S_2 may become frontier edges.

Consider a frontier edge (v_I, v_{E2}), where v_I is the agreed upon label of the interior end in S_m. Assume l is the agreed upon edge ordering of the edge in v_I. Denote the same edge in S_1 (if it exists) by $edge_1\{v_I, l\}$ and the other end vertex of the edge by v_{E1}. Depending on the status of $edge_1\{v_I, l\}$ in S_1, there are a number of possible cases.

- Case 1: (Electronic Fusing)

  $edge_1\{v_I, l\} \in S_1$, $v_{E1} \in S_1$

  Both robot$_1$ and robot$_2$ explored this edge independently (Figure 3.2(a), where S_m



(a) Both the frontier edge and the exterior end were not explored by robot$_1$

(b) Expanding edge (v_I, v_{E2}) into S_m

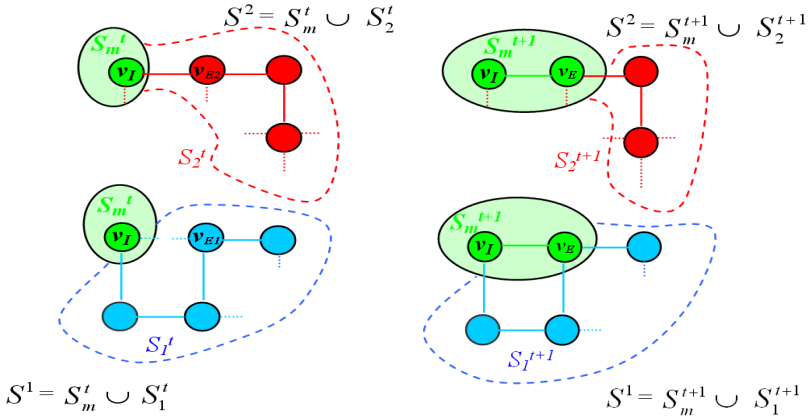
Figure 3.3: Mechanical fusing, marker not found.

is colored green, S_2 colored red and S_1 colored blue). As both robot$_1$ and robot$_2$ have v_I in common, the common ordering of edges in v_I is known. We can also trivially define the common label of v_{E2} and v_{E1}, and the common ordering of the edge at $v_{E2/E1}$ ('fuse' the edges), then expand the 'fused' edge (v_I, $v_{E2/E1}$) into S_m and reduce S_1 and S_2 by the edge (Figure 3.2(b)).

- Case 2: (Mechanical Fusing)

  $edge_1\{v_I, l\} \notin S_1$.

  Robot$_2$ explored this edge but robot$_1$ did not. v_{E2} has been disambiguated against S_2 and S_m but not S_1. Disambiguation against S_1 must be done via mechanical motion of one of the robots. In the simplest case one of the robots moves to v_{E2} and drops its marker. It then visits all potential matches to v_{E2} in S_1. There are two possibilities.



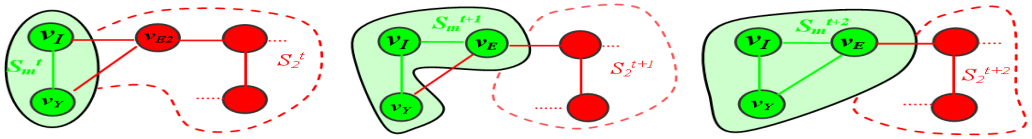
(a) The frontier edge was not explored by robot$_1$, but the exterior end was

(b) Expanding edge (v_I, $v_{E2/E1}$) into S_m

Figure 3.4: Mechanical fusing, marker found.

– Case 2a: The marker is **not** found in S_1. Then clearly vertex $v_{E2} \notin S_1$ (Figure 3.3(a)). We can simply add the edge (v_I, v_{E2}) to S_m and remove the edge from S_2 (Figure 3.3(b)).

– Case 2b: The marker is found in some vertex in S_1 (call this vertex v_{E1}, as in Figure 3.4(a)). In this case v_{E1} and v_{E2} correspond to different local labellings by the two robots of the same vertex. It is necessary to establish the common ordering of the edge (its embedding) in $v_{E2/E1}$ ('fuse' v_{E2} and v_{E1}). There are many possible approaches to this, but perhaps the simplest is to have the robot go back to v_I, drop its marker, move to $v_{E2/E1}$ and then try each edge in $v_{E2/E1}$ to identify the specific edge that leads to v_I. Once this has been accomplished, the edge $(v_I, v_{E2/E1})$ can be added to S_m and removed from S_2. Vertex v_{E1} is removed from S_1 (Figure 3.4(b)).



(a) (v_Y, v_{E2}) is a frontier edge

(b) Establishing (v_I, v_E) makes (v_Y, v_{E2}) a direct edge

(c) Expanding direct edge (v_Y, v_E) into S_m directly

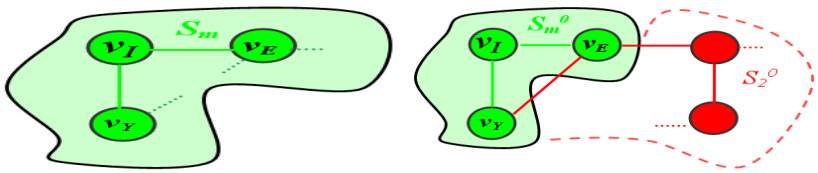
Figure 3.5: Direct edge and direct fusing.

Direct edge and direct fusing The process of establishing the edge $(v_I, v_{E2/E1})$ in S_m can change edges in S_2 in different ways. While the process can change inner edges in S_2 into (new) frontier edges, (eventually) leading to visits of all edges and vertices in S_2, it can also provide evidence of other edges that can be merged directly (electronically). To see this, suppose vertex $v_Y \in S_m$ and edge (v_Y, v_{E2}) is an edge that has been explored by robot$_2$ (the edge may or may not be explored by robot$_1$). Edge (v_Y, v_{E2}) is another frontier edge that has not been processed yet (Figure 3.5(a)). Establishing edge $(v_I, v_{E2/E1})$ makes both ends of edge (v_Y, v_{E2}) commonly known (in S_m), therefore also trivially establishes edge $(v_Y, v_{E2/E1})$ in S_m ('fuses' edge (v_Y, v_{E2}) and the corresponding edge (v_Y, v_{E1}) if the corresponding edge exists in S_1) and allows us to remove the edge (v_Y, v_{E2}) from S_2 (Figure 3.5(b), 3.5(c)). (Also remove the corresponding edge (v_Y, v_{E1}) from S_1 if it exists in S_1.) We call edge $(v_Y, v_{E2/E1})$ a *direct edge* and the establishment of the edge in S_m *direct fusing*. A direct edge is an independently explored edge (by robot$_2$) in S_2 but whose two end vertices are all in S_m. Formally, a direct edge is an edge $e = (v_1, v_2)$ such that

1. $e \in S_2$.

2. $v_1 \in S_m$ and $v_2 \in S_m$.

Direct edges are not only generated during the merge process, but they may also exist before the merge starts. Direct edges that exist before the merge process starts are



(a) New common map for both robots after merge phase n

(b) Results of subsequent exploration phase by robot$_2$ (before merge phase $n+1$)

Figure 3.6: Direct edge existing before merging starts.

generated in the preceding exploration phase in which robot$_2$ augments S^2 by exploring edges connecting vertices in S_m (Figure 3.6).

Each iteration of the algorithm involves processing frontier edges (using 'electronic fusing' and 'mechanical fusing') and direct edges (using 'direct fusing') from S_2. The algorithm terminates when frontier edges and direct edges are exhausted. A summary of the evolvement of the partial maps during the merge process is present in Table 3.1. Note that in this table $S_1^t \cup S_m^t$ becomes the new common map after the merge.

3.2.3 Statement of the algorithm

This section provides a formal description of the fundamental multiple robot exploration algorithm. Striving to be consistent with the notation of the exploration algorithm described in [15], we denote the mapping from a local map S^i that the robot$_i$ is constructing to the real world G by ϕ. $\phi_i(v)$ and $\phi_i(e)$ denote the vertex and edge in the real world G which corresponds to vertex v and edge e in S^i, respectively (v and e are local identifications in S^i). Conversely, $\phi_i^{-1}(v)$ and $\phi_i^{-1}(e)$ denotes the reverse mapping from G to S_i. Note that v and e are identifications in the real world G.

Explored edges are specified by the vertices upon which the edge is incident, as well

Operation		Size change of maps during the merge process				
		S_m^t	S_2^t	S_1^t	$S_2^t \cup S_m^t$	$S_1^t \cup S_m^t$
Electronic fusing		Grows by 1 edge 1 node	Shrinks by 1 edge 1 node	Shrinks by 1 edge 1 node	No change	No change
Mechanical fusing	Marker not found	Grows by 1 edge 1 node	Shrinks by 1 edge 1 node	No change	No change	Grows by 1 edge 1 node
	Marker found	Grows by 1 edge 1 node	Shrinks by 1 edge 1 node	Shrinks by 1 node	No change	Grows by 1 edge
Direct fusing		Grows by 1 edge	Shrinks by 1 edge	No change / Shrinks by 1 edge	No change	Grows by 1 edge / No change

Table 3.1: Evolvement of maps during merge process.

as the two indices of the edge defined by the edge ordering for each of the two vertices (an unexplored edge is specified by its known vertex and the edge ordering for the known vertex). Let $edge_i(v, l)$ indicate the edge incident on vertex v, with local index l with respect to v in S^i; $index_i(e, v)$ indicates the local index of edge e incident on vertex v with respect to v in S^i; $degree_i(v)$ indicates the degree of vertex v in S^i. $otherVertex_i(edge_i(v, l), v)$ indicates the other end of an edge that is incident on v and has local index l in S^i. $label_i(e)$ and $label_i(v)$ indicate the label of edge e and vertex v in S^i respectively, $label_i(v/e) \in \{UNEXPLORED, EXPLORED, COMMON\}$. This labelling is with respect to the labelling maintained by robot i.

To deal with vertex and edge identification and index matching between maps, some conversion information must be maintained for queries. To this end, denote the mapping of vertices from local map S^i to S^j by $\mu_{i\to j}$, e.g., $\mu_{1\to 2}(v_1)$ denotes the vertex in S^2, which corresponds to vertex v_1 in S^1. Again, v_1 is the local identification in S^1. To match local indexes ('door numbers') of counterpart edges incident on counterpart vertices in different maps, we maintain local index 'correspondences' between counterpart vertices in different maps, which is denoted by $\alpha_{i\to j}(v_i)$. The 'correspondence' specifies the relationship between the two local edge orderings of the counterpart vertices. For example, in a planar graph and using a clockwise enumeration of edge ordering, then the

correspondence $\alpha_{i\to j}$ would simply be the shift (rotation) between the two edge orderings. This correspondence in turn can be used to infer (compute) the index of other counterpart edges. Given the correspondence α between two counterpart vertices and one edge index j_1 on one vertex v_1, the counterpart index j_2 at the corresponding vertex v_2 can be inferred by a function, denoted β, i.e., $j_2 = \beta(j_1, \alpha_{1\to 2}(v_1))$. For example, in planar graph and clockwise enumeration, $j_2 = \beta(j_1, \alpha_{1\to 2}(v_1)) = (j_1 + \alpha_{1\to 2}(v_1) + degree_1(v_1))$ *mod* $degree_1(v_1)$, where $\alpha_{1\to 2}(v_1)$ is the local ordering shifting distance.

Algorithm 1 (MERGE(S^1,S^2)).

▷ **this algorithm merges robot$_2$'s map S^2 into base robot$_1$'s base map S^1**
▷ **$S^1 = S_1 \cup S_m$ where S_1 is the unmerged part of map**
▷ **$S^2 = S_2 \cup S_m$ where S_2 is the unmerged part of map**
▷ **S_m is the merged, common part of the maps, but has two 'views' (copies)**
▷ **specification of S_m in S^1 ($S_1 \cup S_m$) consistent with that in S_1 (robot$_1$'s view)**
▷ **specification of S_m in S^2 ($S_2 \cup S_m$) consistent with that in S_2 (robot$_2$'s view)**
▷ **algorithm maintains set C of commonly known vertices**
▷ **C is specified in robot$_2$'s view, initially contains the sole start vertex**
▷ **algorithm maintains set F of frontier edges, specified in robot$_2$'s view**
▷ **algorithm maintains set D of direct edges, specified in robot$_2$'s view**
▷ **algorithm terminates when both set F and D are empty**
▷ **upon termination, $S_1 \cup S_m$ (augmented S^1) is the merged result**

marker := the (unique) marker of the executing robot
▷ **preceding exploration may generate frontier edges and direct edges**
▷ **scan through vertices currently in common set C**
for all vertices v_{common} in C
    for all edges f incident on v_{common}
        if isFrontierEdge (f)
            add f to F ▷ **specification in F is in robot$_2$'s view**
        elseif isDirectEdge (f)
            add f to D ▷ **specification in D is in robot$_2$'s view**
    end for
end for

▷ **main merge loop**

exit when $F = \{\}$ *and* $D = \{\}$

▷ **first try 'electronic fusing' possibility**

for each $e_2 = (v_{I2}, l_2)$ *in* F ▷ v_{I2} **is the common (green) end of** e_2 **in** S_m

▷ **infer frontier edge** e_2**'s counterpart edge** e_1 **in** $S_1 \cup S_m$

▷ **by inferring** v_{I2}**'s counterpart vertex** v_{I1}**, given by** $\mu_{2\to1}(v_{I2})$**, and**

▷ e_1**'s index on** v_{I1}**, given by** $\beta(l_2, \alpha_{2\to1}(v_{I2}))$

$v_{I1} := \mu_{2\to1}(v_{I2})$

$e_1 := edge_1(v_{I1}, \beta(l_2, \alpha_{2\to1}(v_{I2})))$

if $e_1 \in S_1$ ▷ $e_1 \in S_1 \cup S_m$**, the edge was also explored by robot**$_1$

▷ e_1 **and** e_2 **should be fused (merged)**

▷ **first retrieve other end** v_E

$v_{E2} := otherVertex_2(e_2, v_{I2})$ ▷ **in** S_2

$v_{E1} := otherVertex_1(e_1, v_{I1})$ ▷ **in** S_1

▷ **fuse (merge)** e_1 **and** e_2 **electrioncally**

▷ **by recording matching information for** v_{E1} **and** v_{E2}

$\mu_{1\to2}(v_{E1}) := v_{E2}$

$\mu_{2\to1}(v_{E2}) := v_{E1}$

$\alpha_{1\to2}(v_{E1})$ *:= correspondence of* $index_1(e_1, v_{E1})$ *and* $index_2(e_2, v_{E2})$

$\alpha_{2\to1}(v_{E2})$ *:= correspondence of* $index_2(e_2, v_{E2})$ *and* $index_1(e_1, v_{E1})$

▷ **now evolve local maps** S^1 **and** S^2

▷ **for robot**$_1$**'s map (base map)**

$label_1(e_1) = COMMON$ ▷ **'paint green',** S_m **(by robot**$_1$**) grows**

$label_1(v_{E1}) := COMMON$ ▷ **and** S_1 **shrinks**

▷ **for robot**$_2$**'s map**

remove e_2 *from* F ▷ **no longer a frontier edge**

$label_2(e_2) := COMMON$ ▷ **'paint green',** S_m **(by robot**$_2$**) grows**

$label_2(v_{E2}) := COMMON$ ▷ **and** S_2 **shrinks**

add v_{E2} *into* C ▷ **a new common vertex**

▷ v_{E2} **may contribute new frontier edges and direct edges**
for each edge f *emanating* v_{E2}
if isFrontierEdge (f)
add f *to* F
elseif isDirectEdge(f)
add f *to* D
end for
end if
end for ▷ **finish all possible electronic fusing of frontier edges**

▷ **try validate remaining frontier edges by mechanical movement**
▷ **the algorithm is similar to the exploration algorithm in [14, 15]**

▷ **choose a frontier edge** e_2 **from** F
▷ **with common end** v_{I2} **that is closest to** $v_{current}$
▷ $v_{current}$ **is where the robot is currently in**
▷ **move the robot to** v_{I2}**, then traverse along the edge**
▷ **drop the marker at the other end** v_{E2} **in** S_2
edge e_2 := *choose(F)* ▷ $e_2 = (v_{I2}, l_2)$ **with common end vertex** v_{I2}
$v_{E2} := otherVertex_2(e_2, v_{I2})$
walk$(\phi_2(v_{current}), \phi_2(v_{I2}))$ ▷ **go to common end** v_I
followEdge$(\phi_2(e_2))$ ▷ **to other end** $\phi(v_{E2})$
drop$(marker)$
followEdge$(\phi_2(e_2))$ ▷ **back**

▷ **search in** S_1 **for the dropped marker (this will change** $v_{current}$**)**
search$(S_1, markerFound)$

if not markerFound ▷ $\phi_2(v_{E2})$ **is a new vertex to** $S_1 \cup S_m$
walk$(\phi_2(v_{current}), \phi_2(v_{I2}))$ ▷ **go to common end** v_I
followEdge$(\phi_2(e_2))$ ▷ **to other end** $\phi_2(v_{E2})$
pickup$(marker)$

▷ **generate frontier edge e_2's counterpart edge e_1 in $S_1 \cup S_m$,**
▷ **by inferring v_{I2}'s counterpart vertex v_{I1}, given by $\mu_{2\to1}(v_{I2})$, and**
▷ **e_1's index on v_{I1}, given by $\beta(l_2, \alpha_{2\to1}(v_{I2}))$**
▷ **also generate e_1's other end v_{E1}**
▷ **and determine index of e_1 on v_{E1}**
$v_{I1} := \mu_{2\to1}(v_{I2})$
$e_1 := edge_1(v_{I1}, \beta(l_2, \alpha_{2\to1}(v_{I2})))$
v_{E1} *:= new vertex whose label is the last (highest)*
$otherVertex_1(e_1, v_{I1}) := v_{E1}$ ▷ **set other end**
$index_1(e_1, v_{E1}) := index_2(e_2, v_{E2})$ ▷ **free to set, just copy**
$degree_1(v_{E1}) := degree_2(v_{E2})$ ▷ **copy electronically**

for each other edge f leaving $\phi_1(v_{E1})$ ▷ **f is label in real world G**
$index_1(\phi_1^{-1}(f), v_{E1})$:= consistent ordering with respect to e_1
add $\phi_1^{-1}(f)$ to U ▷ **U is maintained by base robot$_1$**
end for

▷ **fuse (merge) e_1 and e_2**
▷ **by recording matching information for v_{E1} and v_{E2}**
$\mu_{1\to2}(v_{E1}) := v_{E2}$ $\mu_{2\to1}(v_{E2}) := v_{E1}$
$\alpha_{1\to2}(v_{E1})$ *:= correspondence of* $index_1(e_1, v_{E1})$ *and* $index_2(e_2, v_{E2})$
$\alpha_{2\to1}(v_{E2})$ *:= correspondence of* $index_2(e_2, v_{E2})$ *and* $index_1(e_1, v_{E1})$

▷ **now evolve local maps S^1 and S^2**
▷ **for robot$_1$'s map (base map)**
remove e_1 from U
add e_1 to $S_1 \cup S_m$ ▷ **base map augmented**
add v_{E1} to $S_1 \cup S_m$ ▷ **by one edge and one node**
$label_1(e_1) := COMMON$ ▷ **'paint green', S_m (by robot$_1$) grows**
$label_1(v_{E1}) := COMMON$ ▷ **and S_1 no change**

▷ **for robot$_2$'s map**
remove e_2 from F
$label_2(e_2) := COMMON$ ▷ **'paint green', S_m (by robot$_2$) grows**
$label_2(v_{E2}) := COMMON$ ▷ **and S_2 shrinks**

add v_{E2} *into* C ▷ **new common vertex**

▷ v_{E2} **may contribute new frontier and direct edges**
for each edge f *emanating* v_{E2}
if isFrontierEdge (f)
add f *to* F
elseif isDirectEdge(f)
add f *to* D
end for
end if (not markerFound)

else ▷ **marker found**
▷ $v_{found} = v_{current}$**,** v_{found} **in** S_1 **corresponds to** v_{E2} **in** S_2
▷ **call** v_{found} **as** v_{E1}
▷ **infer the counterpart edge in** $S_1 \cup S_m$
▷ **and determine its index with respect to** v_{E1}

▷ **generate frontier edge** e_2**'s counterpart edge** e_1 **in** $S_1 \cup S_m$**,**
▷ **by inferring** v_{I2}**'s counterpart vertex** v_{I1}**, given by** $\mu_{2\to1}(v_{I2})$**, and**
▷ e_1**'s index on** v_{I1}**, given by** $\beta(l_2, \alpha_{2\to1}(v_{I2}))$
▷ **also determine index of** e_1 **on** v_{E1}
$v_{I1} := \mu_{2\to1}(v_{I2})$
$e_1 := edge_1(v_{I1}, \beta(l_2, \alpha_{2\to1}(v_{I2})))$
$otherVertex_1(e_1, v_{I1}) := v_{E1}$ ▷ **set the other end**

pickup($marker$)
walk($\phi_1(v_{E1}), \phi_1(v_{I1})$) ▷ **go to common end** v_I
drop($marker$)
walk($\phi_1(v_{I1}), \phi_1(v_{E1})$) ▷ **back**

for each edges f *leaving* v_{E1} ▷f **is local label in** S_1
followEdge($\phi_1(f)$) ▷ **to** $v_{unknown}$
if marker at $v_{unknown}$ *then*
$index_1(e_1, v_{E1}) := index_1(f, v_{E1})$
exit for

end if
end for

▷ **fuse (merge)** e_1 **and** e_2
▷ **by recording matching information for** v_{E1} **and** v_{E2}
$\mu_{1\to 2}(v_{E1}) := v_{E2}$ $\quad$ $\mu_{2\to 1}(v_{E2}) := v_{E1}$
$\alpha_{1\to 2}(v_{E1})$ *:= correspondence of* $index_1(e_1, v_{E1})$ *and* $index_2(e_2, v_{E2})$
$\alpha_{2\to 1}(v_{E2})$ *:= correspondence of* $index_2(e_2, v_{E2})$ *and* $index_1(e_1, v_{E1})$

▷ **evolve local maps** S^1 **and** S^2
▷ **for robot$_1$'s map (base map)**
remove e_1 *from* U
add e_1 *to* $S_1 \cup S_m$ $\quad$ ▷ **base map augmented by one edge**
$label_1(e_1) := COMMON$ $\quad$ ▷ **'paint green',** S_m **(by robot$_1$) grows**
$label_1(v_{E1}) := COMMON$ $\quad$ ▷ **and** S_1 **shrinks**

▷ **for robot$_2$'s map**
remove e_2 *from* F
$label(e_2) := COMMON$ $\quad$ ▷ **'paint green',** S_m **(by robot$_2$) grows**
$label_2(v_{E2}) := COMMON$ $\quad$ ▷ **and** S_2 **shrinks**
add v_{E2} *into* C $\quad$ ▷ **new common vertex**

▷ v_{E2} **may contribute frontier or direct edges**
for each edge f emanating v_{E2}
if isFrontierEdge (f)
add f to F
elseif isDirectEdge(f)
add f to D
end for
pickup($marker$)
end else $\quad$ ▷ **finish marker found case**
▷ **end mechanic fusing (of one frontier edge)**

▷ **lastly do direct fusing for direct edges in** D
for each $e_2 = (v_{E2}, l_2)$ *in* D ▷ **specification in robot$_2$'s view**
▷ **infer direct edge** e_2**'s counterpart edge** e_1 **in robot$_1$'s view of** S_m
▷ **by inferring** v_{E2}**'s counterpart vertex** v_{E1}**, given by** $\mu_{2\to1}(v_{E2})$**, and**
▷ e_1**'s index on the counterpart** v_{E1}**, given by** $\beta(l_2, \alpha_{2\to1}(v_{E2}))$
$v_{E1} := \mu_{2\to1}(v_{E2})$
$e_1 := edge_1(v_{E1}, \beta(l_2, \alpha_{2\to1}(v_{E2})))$ ▷ $index_1(e_1, v_{E1})$ **is** $\beta(l_2, \alpha_{2\to1}(v_{E2}))$

▷ e_1 **may or may not be explored by robot$_1$**
if $e_1 \notin S_1$ ▷ $e_1 \notin S_1 \cup S_m$**, edge not explored by robot$_1$**
$v_{Y2} := otherVertex_2(e_2, v_{E2})$ ▷ **common vertex, robot$_2$'s view**
$v_{Y1} := \mu_{2\to1}(v_{Y2})$ ▷ **must have this matching information**

▷ **set other end** v_{Y1} **and infer index of** e_1 **with respect to** v_{Y1}
$otherVertex_1(e_1, v_{E1}) := v_{Y1}$
$index_1(e_1, v_{Y1}) := \beta(index_2(e_2, v_{Y2}), \alpha_{2\to1}(v_{Y2}))$

▷ **now we have all information about new edge** e_1 **w.r.t** $S_1 \cup S_m$
▷ **time to augment base map!**
remove e_1 *from* U
add e_1 *to* $S_1 \cup S_m$ ▷ **based map augmented by one edge**
$label_1(e_1) := COMMON$ ▷ **'paint green',** S_m **(by robot$_1$) grows**

else ▷ $e_1 \in S_1$**, also explored by robot$_1$**
$label_1(e_1) := COMMON$ ▷ **'paint green',** S_m **grows,** S_1 **shrinks**
end if else

▷ S_m **(by robot$_2$) and** S_2 **evolves anyway**
remove e_2 *from* D
$label_2(e_2) := COMMON$ ▷ **'paint green',** S_m **grows,** S_2 **shrinks**
end for ▷ **finish direct fusing**

end main loop
END OF MERGE ALGORITHM

Robot Actions:

followEdge(e)

▷ **The robot takes edge e out of the current vertex**

drop($marker$)

▷ **The robot drops the specified marker at its current location**

pickup($marker$)

▷ **The robot picks up the specified marker at its current location**

subroutines:

isFrontierEdge($edge_2(v_{I2}, l_2)$)

$v_{E2} := otherVertex_2(edge_2(v_{I2}, l_2), v_{I2})$

true if the following conditions are satisfied:

1. $label_2(v_{I2}) := COMMON$

2. $label_2(edge_2(v_{I2}, l_2)) := EXPLORED$ ▷ **not common**

3. $label_2(v_{E2}) := EXPLORED$ ▷ v_{E2} **not common**

isDirectEdge($edge_2(v_{E2}, l_2)$)

$v_{Y2} := otherVertex_2(edge_2(v_{E2}, l_2), v_{E2})$

true if the following conditions are satisfied:

1. $label_2(v_{E2}) := COMMON$

2. $label_2(edge_2(v_{E2}, l_2)) := EXPLORED$ ▷ **not common**

3. $label_2(v_{Y2}) := COMMON$ ▷ **both ends are common**

choose(F)

▷ **choose from edge set F an edge e with common**

▷ **known incident vertex v_1 that is closest to $v_{current}$**

run shortestPath($v_{current}$) with source vertex $v_{current}$,

finds the edge satisfying the above description

walk(v_{from}, v_{to})

run shortestPath(v_{from}) to get shortest path $(e_1, e_2, ..., e_k)$

from v_{from} to v_{to} through S

for i from 1 to k

followEdge($\phi(e_i)$)

end for

search(S, markerFound)

▷ **collect vertices that need to be searched**

build a list l of vertices in S that satisfy:

1. the vertex has unexplored edge(s) incident on it

2. the vertex has the same signature (degree) as the vertex to be disambiguated

▷ **search until marker is found or no where to search**

markerFound := false

loop

exit when markerFound OR list l = {}

run shortestPath($v_{current}$) to get shortest path from $v_{current}$ to all in l

select vertex v_{pcn} in l that is the closest to $v_{current}$, remove v_{pcn} from l

walk($v_{current}$, v_{pcn})

$v_{current}$:= v_{pcn}

if marker is sensed

markerFound := true

pickup(marker)

end if

end loop

shortestPath(source)

run Dijkstra's algorithm to get the shortest path

from source to all other vertices in the graph.

3.2.4 Correctness proof of the merge algorithm

Suppose two robots explore a graph G and define robot$_1$ as the base robot. After an exploration step, we have the base robot's partial map representation S^1 ($S^1 = S_1 \cup S_m$), and the other robot's partial map representation S^2 ($S^2 = S_2 \cup S_m$). We merge S^2 into S^1. The merge process keeps on augmenting S^1 by adding vertices and edges from S^2. We wish to prove that after the merge, the augmented S^1 is (still) isomorphic to a subgraph

of the world model G. In addition, all edges and vertices of S^2 that corresponded to parts of the world that were not in S^1 before the merge process have been added into S^1, i.e. the other robot's exploration efforts have been fully and correctly exploited.

As in [14, 15], we use an extended definition of graph isomorphism, i.e. graphs G and H are said to be isomorphic if and only if they are isomorphic under the usual definition of graph isomorphism, and, in addition, for each vertex v of G and each edge $e = (v, v')$

$$index(e, v) = relabelling(index(\phi(v), \phi(v'), \phi(v)))$$

or simply

$$index(e, v) = relabelling(index(\phi(e), \phi(v)))$$

where ϕ denotes the mapping from the map G to the other map H, i.e., $\phi(v)$ and $\phi(e)$ denote the vertex and edge in H which corresponds to v and e in G, respectively. *relablling* is a permutation of the edge indices at vertex $\phi(v)$ that preserves the edge ordering used by the algorithm, but ensures a common reference edge for the ordering (e.g. clockwise in a planar graph embedding). This simply states that the edges leaving v and $\phi(v)$ have the same labeling, but may be labeled starting from a different reference edge. This condition is referred to as the "edge-index condition". To facilitate the proof, we first revisit and clarify some issues related to the edge-index condition and edge properties.

Property#1 Suppose v and v' refer to the same vertex in the real world in the maps of the two robots. Then each map preserves the edge-index condition with respect to vertices in the real world. If we know the '*correspondence*' (relations of indices) for one pair of their counterpart edges, then for any edge incident on v, counterpart edge incident on v' can be inferred, and the inferred index holds the "edge-index condition".

Justification Since v and v' preserve the edge-index condition,

$$index(e, v) = relabelling(index(\phi(e), \phi(v)))$$
$$index(e', v') = relabelling(index(\phi(e'), \phi(v')))$$

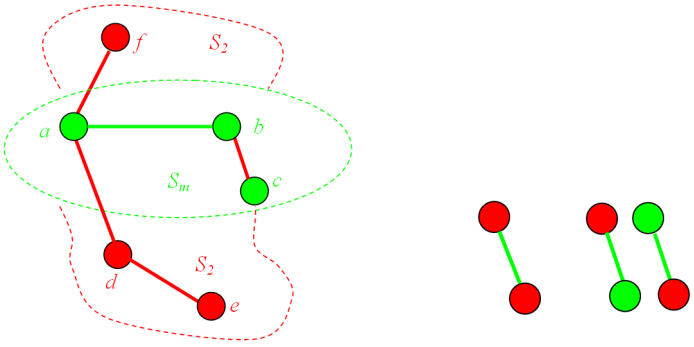
Since v and v' are counterpart vertices, $\phi(v) = \phi(v')$ and $\phi(e) = \phi(e')$. This means that the edge labelling in v and v' are permutations of the same edge labelling (in $\phi(v)$), so the edge labeling of v and v' are permutations of each other too and the edge-index condition is also preserved between v and v', i.e., edges leaving v and v' have the same labelling but start from (potentially) a different reference edge. If we know the correspondence of one pair of edges this establishes the (universal) correspondence of all counterpart edges. So for an edge, based on the correspondence, its counterpart edge is inferable and the inferred index holds the "edge-index condition".

Property#2 At any point during the merge process, in terms of the nature (in S_m or S_2) of edges and their two ends, there are four kinds of edges in S^2 ($S^2 = S_2 \cup S_m$):

1. Inner edges in S_m ('all green' edge ab in Figure 3.7(a)).
2. Inner edges in S_2 ('all red' edge de in Figure 3.7(a)).
3. Frontier edges across S_m and S_2 (edge ad, af in Figure 3.7(a)).
4. Direct edges (edge bc in Figure 3.7(a)).

Justification In terms of the status of an edge $e = (v_1,v_2)$ and the two ends, there are eight combinations and the resulting edges can be arranged into six groups.

1. $e \in S_m$, $v_1 \in S_m$, $v_2 \in S_m$ ('all green' inner edges in S_m)
2. $e \in S_2$, $v_1 \in S_2$, $v_2 \in S_2$ ('all red' inner edges in S_2)



(a) The five kinds of edges existing during the process

(b) The three invalid kinds of edges

Figure 3.7: Existing and non-existing kinds of edges during the merge process.

3. $e \in S_2$, $v_1/v_2 \in S_m$, $v_2/v_1 \in S_2$ (frontier edges)

   (a) $v_1 \in S_m$, $v_2 \in S_2$

   (b) $v_1 \in S_2$, $v_2 \in S_m$

4. $e \in S_2$, $v_1 \in S_m$, $v_2 \in S_m$ (direct edges)

5. $e \in S_m$, $v_1 \in S_2$, $v_2 \in S_2$ (invalid edges)

6. $e \in S_m$, $v_1/v_2 \in S_m$, $v_2/v_1 \in S_2$ (invalid edges)

   (a) $v_1 \in S_m$, $v_2 \in S_2$

   (b) $v_1 \in S_2$, $v_2 \in S_m$

Trivially the first four kinds of edges exist during the merge process. The latter two kinds of edges are common (green) edges with two uncommon end vertices and common (green) edges with one common vertex and one uncommon vertex (Figure 3.7(b)). We show that during the merge process, the latter two kinds of edges are not generated. During the merge process, a common (green) edge either comes from a uncommon (red)

frontier edge when it is disambiguated (electronic fusing or mechanical fusing), or comes from a uncommon (red) direct edge when it is disambiguated (direct fusing). When a frontier edge is disambiguated and fused, the algorithm always includes the frontier edge (red) and its uncommon (red) end into the common area. So both the edge and its uncommon end become common (green). Since the other end of the frontier edge is in the common area (green), the inclusion of the frontier edge makes it a 'all green' edge. When a direct edge is disambiguated and fused, we always include the direct edge (red) into the common area. Since the direct edge has two common (green) ends, the inclusion of the direct edge makes it a 'all green' edge. So a common (green) edge cannot have one or two uncommon red ends and therefore the edges shown in Figure 3.7(b) are not generated. We can therefore infer that:

1. A common (green) edge must have two common (green) ends. This is the only kind of common (green) edge during the merge process.

2. An edge having at least one uncommon (red) end must itself is uncommon (red).

Formal proof of the merge algorithm

We prove the algorithm correct by establishing an invariant **I** and showing that **I** is initially true, is maintained true throughout the execution, and that the algorithm terminates. Then we show that the termination condition plus the invariant imply that after merging, S^1 ($S^1 = S_1 \cup S_m$) is (still) isomorphic to a subgraph of G. In addition, all edges and vertices independently explored by robot$_2$ (edges and vertices of S_2^0) have been disambiguated and merged into S^1, i.e., the other robot's exploration efforts have been fully exploited and utilized.

We define the loop invariant **I** as follows:

I:

1. $S^1 = S_1 \cup S_m$ is isomorphic to subgraph of G, and
2. We maintain a common vertex set C that contains all of the vertices in S^2 that have been disambiguated. We also maintain matching information for all vertices in C.
3. We maintain a frontier edge set F that contains all frontier edges that emanate from the vertices currently in C.
4. We maintain a direct edge set D that contains all direct edges that connects the vertices currently in C.

We also define a bound function, t, for the loop

$$t = |E_{S_2}|$$

where E_{S_2} is the set of edges in S_2 (specifically S_2^t), i.e., the (red) edges in S^2 that have not been disambiguated and merged. $|E_{S_2}|$ denotes the cardinality of the set E_{S_2}.

1. I is true before the loop is entered. Before the merge loop starts, $S^1 = S_1 \cup S_m$ is the original exploration result of the base robot in the preceding exploration phase. The correctness of the exploration algorithm guarantees that $S_1 \cup S_m$ is isomorphic to a subgraph H of G [14, 15], so **I-1** is true. Before the first merge starts, the common vertex set C contains initial common starting point $v_{initial}$. The algorithm also maintains the matching information for $v_{initial}$. This is trivially possible because all robots use the same vertex label and index information (e.g. they treat $v_{initial}$ as same 0'th vertex in both local maps and same edge ordering in $v_{initial}$). This establishes invariant **I-2**. If the merge is not the first merge, the common vertices in C are vertices of the merged map from the last merge. The algorithm maintains the matching information for the common vertices. This is possible because all robots use the same vertex label and index information. This also maintains the invariant **I-2.** Also, all edges emanating from vertices in C are checked,

those that satisfy the frontier edge definition are added to F. Thus F contains all frontier edges emanating current vertices in C, maintaining invariant **I-3**. All edges emanating vertices in C are checked, those who satisfy the direct edge definition are added to D, thus D contains all direct edges emanating current vertices in C, maintaining invariant **I-4**.

2. I is maintained by the loop body. Each time through the loop body, frontier edges in F and direct edges in D are processed if they exist. The processing may add new edges into $S_1 \cup S_m$. We show that in each iteration, if edges are added, each added edge is correctly labeled (indexed) with respect to its two ends in $S_1 \cup S_m$, maintaining **I-1**. In addition, C, F and D are correctly updated, maintaining **I-2**, **I-3** and **I-4**. The algorithm first processes frontier edges. As elaborated before, depending on the status of the current selected frontier edge in robot$_1$'s map there are 3 different execution possibilities.

Case 1 First, the algorithm scans through all current frontier edges in F, seeking electronic fusing possibilities. For each frontier edge e_2, the algorithm computes e_2's counterpart edge in $S_1 \cup S_m$. This computation is possible, since by the hypothesis that **I-2** is true before this iteration, the algorithm maintained in all common vertices in C their correct matching information, and **I-3** states that each (frontier) edge in F is an edge emanating from an vertex in C. This computation is also correct, as justified in **Property#1**.

If the computed edge (call it e_1) exists in $S_1 \cup S_m$ (specifically, exists in S_1), we add e_2's other end v_{E2} (the exterior vertex in S_2) into common area S_m, recording the matching information for v_{E2} and its counterpart vertex. The matching information is computable because e_1 is a known edge in $S_1 \cup S_m$, by examining the maps, we know the index of e_1 incident on v_{E1}, which is the counterpart vertex of v_{E2}. We can infer the correspondence of the edge index between v_{E1} and v_{E2}. The label conversion information (v_{E1} vs. v_{E2})

and the edge index correspondence between them define the full matching information of the two counterpart vertices. Thus **I-2** is maintained. Including v_{E2} into the common area S_m may generate both frontier edges and direct edges, so the algorithm examines all emanating edges incident on v_{E2}, and updates F and D accordingly. Therefore after the Case 1, both C, D and F are updated, maintaining **I-2, I-3** and **I-4**. During electronic fusing, $S_1 \cup S_m$ is not augmented, so **I-1** is trivially maintained as well.

After processing all of the current frontier edges that can be electronically fused, the algorithm checks F again. If there is a frontier edge left, it will be processed via mechanical moves. This is essentially an exploration by the base robot on the two maps. The algorithm drops a marker at the uncommon (exterior) other end of the frontier edge, say, v_{E2} in S_2, search in S_1 to see if the marker can be found. Since v_{E2} is a vertex in S_2, the correctness and completeness of searching in S_1, rather than $S_1 \cup S_m$ is given by the fact that both before and during the merge process, vertices in both S_1 and S_2 are distinct with that in S_m. The algorithm performs one of the following mechanical steps, depending on the outcome of the marker search process.

Case 2a In the case that the marker is not found in S_1, then both the counterpart edge of the frontier edge and the counterpart vertex of the frontier edge's uncommon end are not in S_1, the algorithm adds a new edge $e_1 = (v_{I1}, v_{E1})$ to $S_1 \cup S_m$, where v_{I1} is the vertex in S_m that is the counterpart vertex of the frontier edge's common end v_{I2} and v_{E1} is the counterpart vertex of the frontier edge's uncommon end v_{E2}. By **I-2**, **Property#1** and **I-1**, $index(e_1, v_{I1})$ is correct and satisfy the edge index condition. The algorithm is free to set $index(e_1, v_{E1})$ arbitrarily, since no indices of edges leaving v_{E1} have yet been set. The algorithm uses the index of the frontier edge e_2 with respect to its end v_{E2}, i.e., $index(e_1, v_{E1}) = index(e_2, v_{E2})$, which, by the correctness of exploration algorithm [14, 15], as well as **I-1** maintained, holds the edge-index condition. So e_1 is correctly labeled, therefore $S_1 \cup S_m$ is updated correctly, maintaining **I-1**. The algorithm

adds v_{E2} to C and records matching information for v_{E2} and v_{E1}, maintaining **I-2**. This information is possible and correct, by **I-2** and **Property#1**. In addition, the algorithm checks edges leaving v_{E2}, for possible new frontier edges and direct edges, and updates F and D accordingly, maintaining **I-3** and **I-4**.

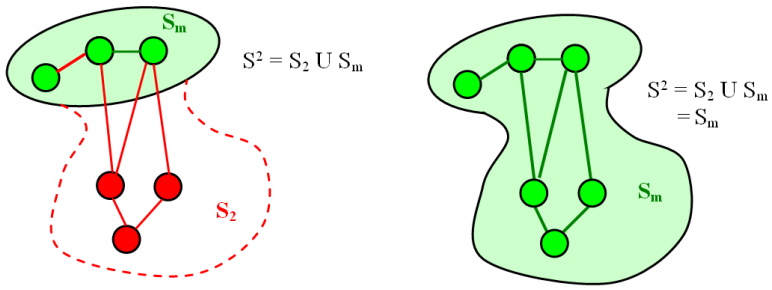
Case 2b In the case that the marker is found, say, at v_{E1} in S_1, then v_{E1} is the counterpart vertex of the frontier edge's (uncommon) exterior end v_{E2}. The algorithm adds a new edge $e_1 = (v_{I1},\ v_{E1})$ in $S_1 \cup S_m$, where v_{I1} is the counterpart vertex of the common end of the frontier edge. Again, by **I-2** , **Property#1** and **I-1**, $index(e_1, v_{I1})$ is correct and satisfies the edge index condition. In the loop body, a search is conducted to determine which of the edges leaving v_{E1} is actually e_1. The search is guaranteed to terminate successfully because both v_{I1} and v_{E1} are in $S_1 \cup S_m$ and thus, e_1 must be in $S_1 \cup S_m$. Furthermore, by hypothesis **I-1** that $S_1 \cup S_m$ is isomorphic to subgraph in G, e_1 will be correctly labeled with respect to v_{E1} when it is determined which edge is e_1, i.e. $index(e_1, v_{E1})$ is correct and satisfies the edge index condition. So $S_1 \cup S_m$ is updated correctly, maintaining **I-1**. Then the algorithm adds v_{E2} to C and records matching information for v_{E2} and v_{E1}. Again, this information is obtainable and correct, by **I-2** and **Property#1**, thus maintaining **I-2**. Also, the algorithm checks emanating edges of v_{E2} for possible new frontier edges and direct edges and updates F and D, maintaining **I-3** and **I-4**.

Direct fusing After performing possible mechanical fusing (for one frontier edge), the algorithm performs direct fusing for each edge in the current D. As stated earlier, in direct fusing, for each direct edge $e_2 = (v_{E2},\ v_{Y2})$ (in robot$_2$'s view), the algorithm computes the direct edge's counterpart edge e_1 in robot$_1$'s view (the two counterpart vertices v_{E1} and v_{Y1} along with the counterpart edge's index with respect to the two counterpart ends). The computation is possible, since for the two common ends we have correctly recorded the matching information, by the hypothesis that **I-2** is true. The computation

is correct, as **Property#1** shows. So the computed edge is correctly labeled (indexed) with respect to its two ends in robot$_1$'s view, being consistent with $S_1 \cup S_m$. So we can check if the computed edge exists in $S_1 \cup S_m$ (specifically, if it exists in S_1). If not, we add it to $S_1 \cup S_m$, using the computed vertex and index information which is consistent with the labeling in $S_1 \cup S_m$. So $S_1 \cup S_m$ is updated by a new edge that satisfies the edge index condition. Thus **I-1** is also maintained. This process does not generate new common vertices, consequently no new frontier edges and direct edges are generated and C, F and D are unchanged and therefore **I-2, I-3** and **I-4** are maintained.

So, in each iteration of the loop body, **I-1** is maintained by updating $S_1 \cup S_m$ with correctly labeled edge(s) (consistent with $S_1 \cup S_m$); **I-2** is maintained by updating C with new common vertices and recording conversion information at the new common member vertices. **I-3** is maintained by adding to F all frontier edges emanating vertices in C, **I-4** is maintained by updating D with all direct edges emanating vertices in C.

3 The loop terminates. The loop continues while frontier edges or direct edges remain. By definition, frontier edges and direct edges are edges of S_2, which is the unmerged part of map S^2. If frontier edges exist, then in each loop iteration, the algorithm chooses and processes at least one frontier edge from F and then depending on the status of the frontier edge's counterpart edge in $S_1 \cup S_m$, the frontier edge may be electronically fused or mechanically fused. At the end of execution the current frontier edge is always included in S_m, becoming a new common (green) edge in S_m. Since frontier edges are edges of S_2, we are actually decreasing at least one edge from S_2, i.e., decreasing one element in E_{S_2}. Similarly, if direct edges exist, the (red) direct edge will be turned into common (green) edges, also decreasing at least one element in E_{S_2}. So in each iteration (pass) of the loop body, the bound function t is decreased. The bound function t is non-negative. So the loop must terminate eventually, since t can only remain non-negative for a finite number



(a) S^2 before merge (b) S^2 after merge

Figure 3.8: Change of S_2 and S_m in S^2 before and after merge.

of iterations (passes) through the loop.

4 When the loop terminates, $S_1 \cup S_m$ is (still) isomorphic to subgraph of G, and all independently explored edges and vertices in S^2 have been disambiguated and merged. By maintaining **I-1** and **I-2**, when a frontier edge and its other end is visited, if it does not exist in $S_1 \cup S_m$, $S_1 \cup S_m$ will be updated correctly, so when the loop terminates, $S_1 \cup S_m$ will be updated with correctly labeled edges. When a direct edge is visited, if it does not exist in $S_1 \cup S_m$ then $S_1 \cup S_m$ will be updated correctly. Hence $S_1 \cup S_m$ is (still) isomorphic to a subgraph of G.

We also show that when the loop terminates, all independently explored edges and vertices in the original S^2 have been disambiguated and merged (integrated into the common area S_m), i.e., upon termination, $S_2 = \{\}$, $S^2 = S_2 \cup S_m = S_m$ (unmerged part S_2 disappears – S^2 becomes 'all green' as shown in Figure 3.8). Assume, to the contrary, that when the loop terminates ($F = \{\}$ and $D = \{\}$) there are some unmerged (red) edges in S^2, i.e., $S_2 \neq \{\}$. Then from **Property#2**, we can see that an unmerged (red) edge can be a frontier edge, a direct edge and inner edge in S_2 (with both two red ends).

Since F and D are empty, it can only be inner edges in S_2, i.e., red edge with both two red ends. Now assume v_{red} is one of the (red) ends. Then since $S_2 \cup S_m$ is connected, there must be a path from $v_{initial}$ (green) to v_{red}. This requires that there be an edge on this path with one green end, and one red end. **Property#2** infers that this can only be a red edge, which, by the definition and property of frontier edge, is a good frontier edge. By **I-3**, this edge must be in F, contradicting the terminating condition that $F = \{\}$. So there can be no unmerged (uncommon) edges in $S^2 = S_2 \cup S_m$, i.e., all edges are common (green). **Property#2** also infers that common green edges must have green ends. Therefore, we conclude that there are no unmerged (red) edges and unmerged (red) vertices in $S^2 = S_2 \cup S_m$, i.e., all edges and vertices in $S^2 = S_2 \cup S_m$ have been processed. **I-1** guarantees that once processed, the edge or vertex is correctly handled.

By the loop invariant **I** and the termination condition, we thus conclude that the algorithm terminates with the base map $S^1 = S_1 \cup S_m$ being (still) isomorphic to a subgraph of G and all efforts of the other robot have been fully exploited (Figure 3.8(a)).

3.3 Evaluation Mechanism

To compare different solutions presented in this report a performance metric for generic multiple robot exploration is required. Our main performance concern is the mechanical cost of completing the exploration task.

Measuring work in a parallel system In a parallel system, there may exist both parallel and sequential work, due to possible synchronization mechanisms. For example, in a parallel system, the working agents may stop after a certain amount of independent parallel work, when all agents have finished the work and stopped, the agents start working in parallel again (Figure 3.9). For such situations, if we want to measure the resource cost (e.g., time spent, distance traveled) in the parallel system, we need to divide

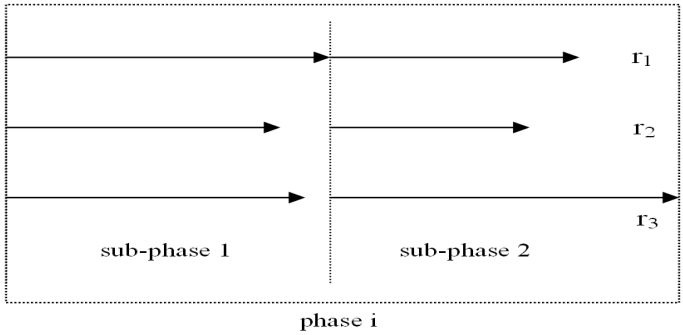


Figure 3.9: A parallel phase divided into two subphases.

the system into subphases according to the synchronization mechanisms used so that work in each subphase is completely parallel. Then for each parallel subphase, we can take the maximum cost (e.g., the effort involved) by the parallel work as the cost of this subphase. Subphases are sequential, so we take the sum of cost of them as the cost of the whole phase. So for the example shown in Figure 3.9, the total cost of the whole phase is the sum of the maximum cost in subphase$_1$ and the maximum cost in subphase$_2$.

Parallelism in multiple robot exploration In real life multiple robots can explore independently and concurrently. In evaluating the multiple robot exploration algorithm we depart slightly from the earlier model and assume that robots can operate in complete parallel. So in multiple robot exploration, there exists both parallel work and sequential work. This is illustrated in Figure 3.10, which shows that the exploration algorithm can be decomposed into multiple (parallel) phases ($E_{11}, E_{12}, ..., M_{nm}$). During each phase, each robot is completely independent and operate in complete parallel. The parallel phases are separated by periods in which the robots are completely synchronized.

Task cost Here we formally define the cost for the generic multiple robot exploration algorithm. Suppose that each exploration and merge phase is divided into one or more subphases in such a way that work in each subphase is completely parallel. Let E_i denote

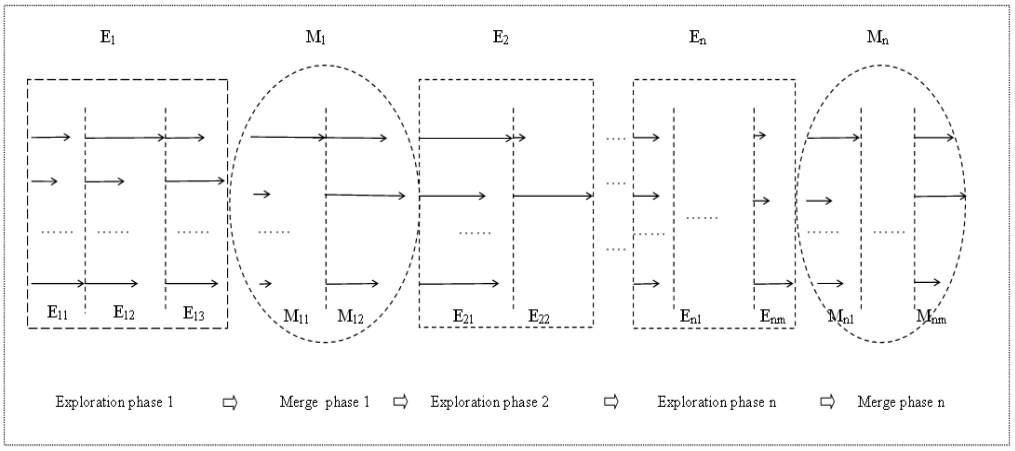


Figure 3.10: Alternating phases of exploration and merge.

exploration phase i and let M_i denote merge phase i; For an exploration phase E_i that has m ($m \geq 1$) subphases, denote subphase j ($1 \leq j \leq m$) in E_i as E_{ij}. Similarly, for a merge phase M_i that has m ($m \geq 1$) subphases, denote subphase j ($1 \leq j \leq m$) in M_i as M_{ij}. Let $r_l(E_{ij})$ denote the mechanical cost (number of edge traversals) taken by robot l in subphase E_{ij}; Let $r_l(M_{ij})$ denote the mechanical cost taken by robot l in merge phase M_{ij}. Denote the cost for exploration phase E_i and its subphase E_{ij} as $cost(E_i)$ and $cost(E_{ij})$ respectively. Denote the cost for merge phase M_i and its subphase M_{ij} as $cost(M_i)$ and $cost(M_{ij})$.

We start from the complete parallel work in each subphase. For each subphase E_{ij}, the cost of the subphase is the maximum mechanical cost among all robots, i.e.,

$$cost(E_{ij}) = \max_{l=1}^{k}\{r_l(E_{ij})\}.$$

Subphases in E_i are sequentially separated, so the cost of the exploration phase E_i is the

sum of the cost of all its m subphases, i.e.,

$$\begin{aligned} cost(E_i) &= \sum_{j=1}^{m} cost(E_{ij}) \\ &= \sum_{j=1}^{m} \max_{l=1}^{k} \{r_l(E_{ij})\}. \end{aligned}$$

Similarly, for a merge phase M_i having m subphases, the cost in each subphase M_{ij} is given by

$$cost(M_{ij}) = \max_{l=1}^{k} \{r_l(M_{ij})\}.$$

and the cost of phase M_i is given by

$$\begin{aligned} cost(M_i) &= \sum_{j=1}^{m} cost(M_{ij}) \\ &= \sum_{j=1}^{m} \max_{l=1}^{k} \{r_l(M_{ij})\}. \end{aligned}$$

The alternating phases of exploration and merge in the overall exploration process occur sequentially, so the total cost required to complete the task, denoted $TaskCost$, is then given by

$$TaskCost = \sum_{i=1}^{n} (cost(E_i) + cost(M_i)). \tag{3.1}$$

Task cost in the basic algorithms We assume that in the basic multiple robot exploration algorithm described above, the work in each exploration phase is conducted in complete parallel and there is no coordination among the robots. Therefore the cost for a exploration phase is the maximum mechanical work among the robots in the whole phase. Formally, each exploration phase E_i has $m = 1$ subphase (the phase itself), the

cost of phase E_i for the collection of k robots is given by

$$\begin{aligned} cost(E_i) &= \sum_{j=1}^{m} \max_{l=1}^{k}\{r_l(E_{ij})\} \\ &= \max_{l=1}^{k}\{r_l(E_i)\}. \end{aligned}$$

In the basic multiple robot exploration algorithm, the work in each merge phase is conducted by the base robot. Trivially, there is one subphase for each M_i. Moreover, since only the base robot performs the merge task, the cost of each merge phase is the cost of the base robot in the whole merge phase, i.e.,

$$\begin{aligned} cost(M_i) &= \sum_{j=1}^{m} \max_{l=1}^{k}\{r_l(M_{ij})\} \\ &= \max_{l=1}^{k}\{r_l(M_i)\} \\ &= r_{base}(M_i). \end{aligned}$$

The task cost for the basic multiple robot exploration algorithm, is therefore given by

$$\begin{aligned} TaskCost &= \sum_{i=1}^{n}(cost(E_i) + cost(M_i)) \\ &= \sum_{i=1}^{n}(\max_{l=1}^{k}\{r_l(E_i)\} + r_{base}(M_i)). \end{aligned} \tag{3.2}$$

Note that the task cost for the single robot exploration algorithm ([14, 15]) is trivially the mechanical cost of the (sole) robot during the (sole) exploration phase.

3.4 Sample operations

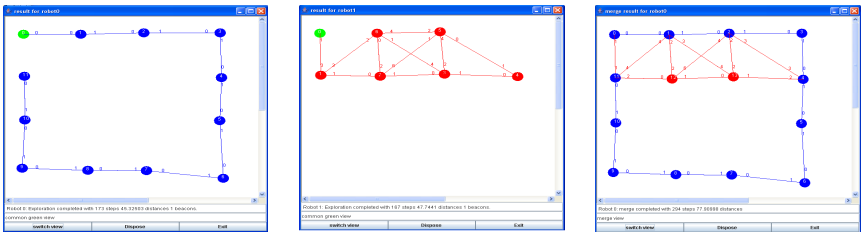
In this section we present sample solutions to the graph mapping problem for two robots. We first demonstrate the algorithm on a simple graph, then we show its operation on a

graph simulating a real environment – the Beijing subway system. Finally we evaluate the performance of the algorithm on different graphs, and compare the performance with the single robot exploration algorithm on the same graphs.

Operation on a lattice-like graph

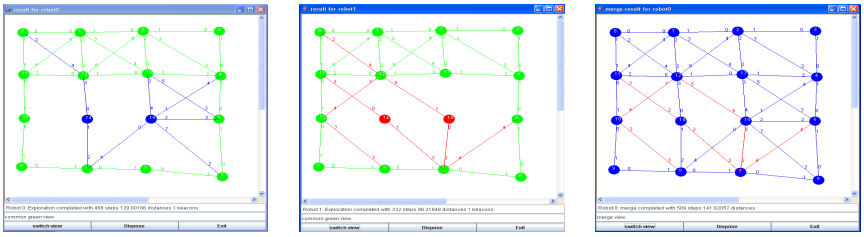
First we demonstrate the algorithm on the lattice-like graph. The algorithm proceeds with alternating phases of independent exploration and coordinated merging of the independently acquired partial representations. In this example both of the two robots start at the upper left corner vertex (vertex 0). This is the initial common map shared by the robots. The rendezvous schedule is 160 mechanical steps and rendezvous occurs at vertex 0, i.e., each of the robots explores for 160 mechanical steps and then returns to vertex 0 to merge their world representations. We designate robot_1 as the base robot, whose partial map is augmented. The merged map then becomes the new common map and the unknown portions of the new common map are partitioned. The partitioning is performed on unknown edges of the new common map, and the unknown edges are assigned to the two robots evenly but in an arbitrary fashion. After the merging and partitioning, a new phase of independent exploration starts, in which the new common maps are augmented independently by the robots. This example involves three phases of independent exploration and three phases of coordinated merging, with the last (third) merge phase generating the full map of the world. Exploration and merge results of each of the phases are shown in Figures 3.11. As in earlier illustrations the common part of the map (the merge result of the last merge phase) is colored green, the independently explored part of robot_1's map is colored blue and the independently explored part of robot_2's map is colored red. For the merged maps, vertices and edges from robot_1's map (the original base map) are colored blue, and edges and vertices newly augmented (from robot_2's map) are colored red.

Exploration phase 1 (160 exploration steps) and merge phase 1



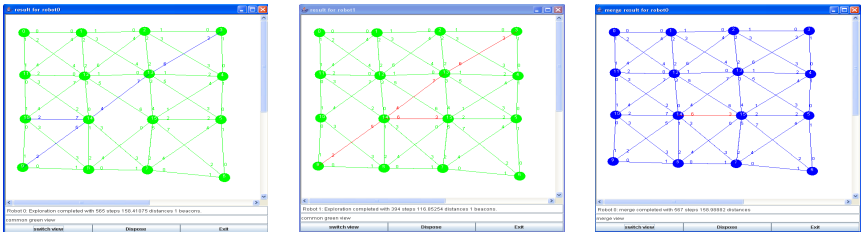
(a) Robot$_1$'s map (b) Robot$_2$'s map (c) Merged map

Exploration phase 2 (320 exploration steps) and merge phase 2



(d) Robot$_1$'s map (e) Robot$_2$'s map (f) Merged map

Exploration phase 3 (480 exploration steps) and merge phase 3
The full map is generated and the algorithm terminates



(g) Robot$_1$'s map (h) Robot$_2$'s map (i) Merged map

Figure 3.11: Multiple robot exploration on lattice-like graph, starting at vertex 0.

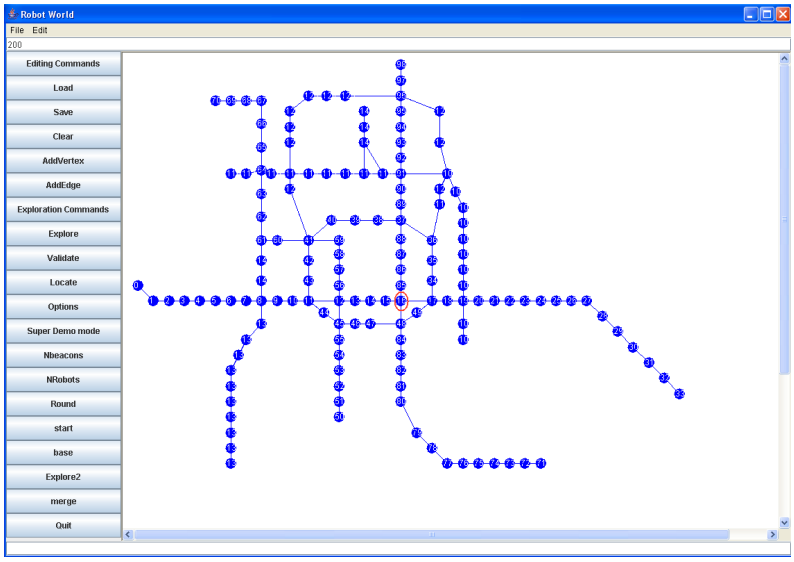
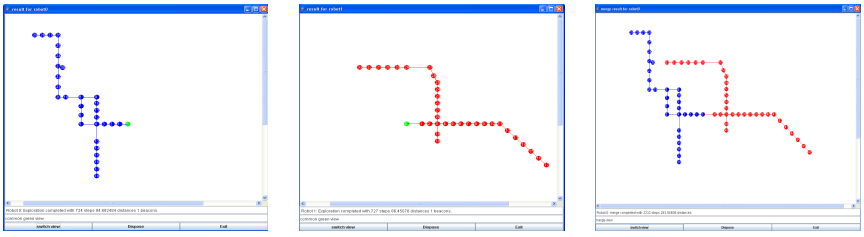


Figure 3.12: Graph simulating Beijing subway system.

Operation on a graph simulating a real system

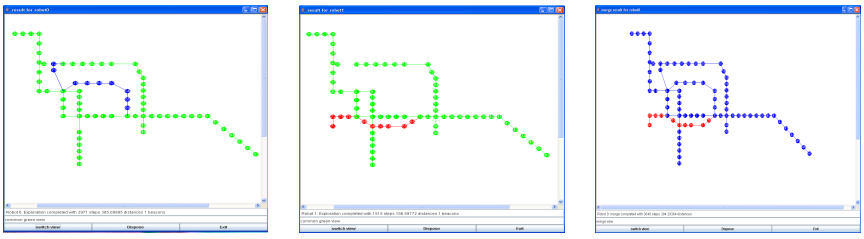
Here we demonstrate the algorithm on the Beijing subway system. Vertices in the graph simulating the system represent subway stations and edges in the graph represent tunnels connecting the stations (Figure 3.12). In this example both of the two robots start at an interchange station ('*DongDan Station*'), which is the initial common map shared by the robots. This is represented by vertex 16 (colored red) in the graph. The rendezvous schedule is 700 mechanical steps and vertex 16 is the rendezvous location. Robot_1 is designated as the base robot. The partition strategy used here is the same as that used in the previous example. This example involves five phases of independent exploration and five phases of coordinated merging. Exploration and merge results from the alternating phases are shown in Figures 3.13–3.14.

Exploration phase 1 (700 exploration steps) and merge phase 1



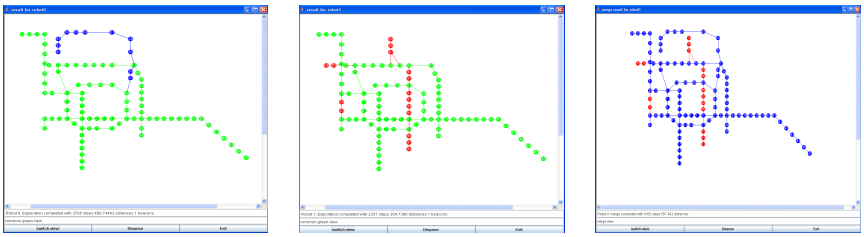
(a) Robot$_1$'s map (b) Robot$_2$'s map (c) Merged map

Exploration phase 2 (700×2 exploration steps) and merge phase 2



(d) Robot$_1$'s map (e) Robot$_2$'s map (f) Merged map

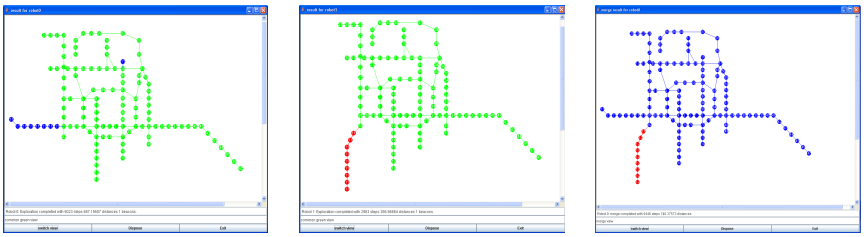
Exploration phase 3 (700×3 exploration steps) and merge phase 3



(g) Robot$_1$'s map (h) Robot$_2$'s map (i) Merged map

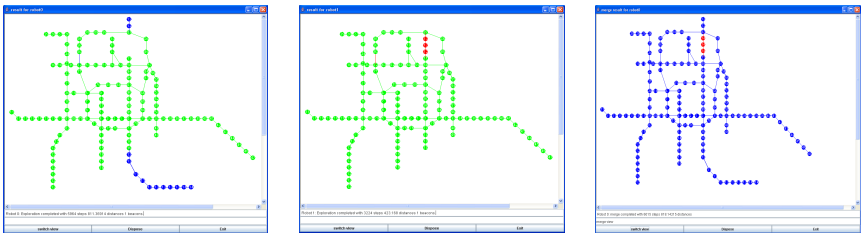
Figure 3.13: Multiple robot exploration on real system, phase 1–3.

Exploration phase 4 (700×4 exploration steps) and merge phase 4



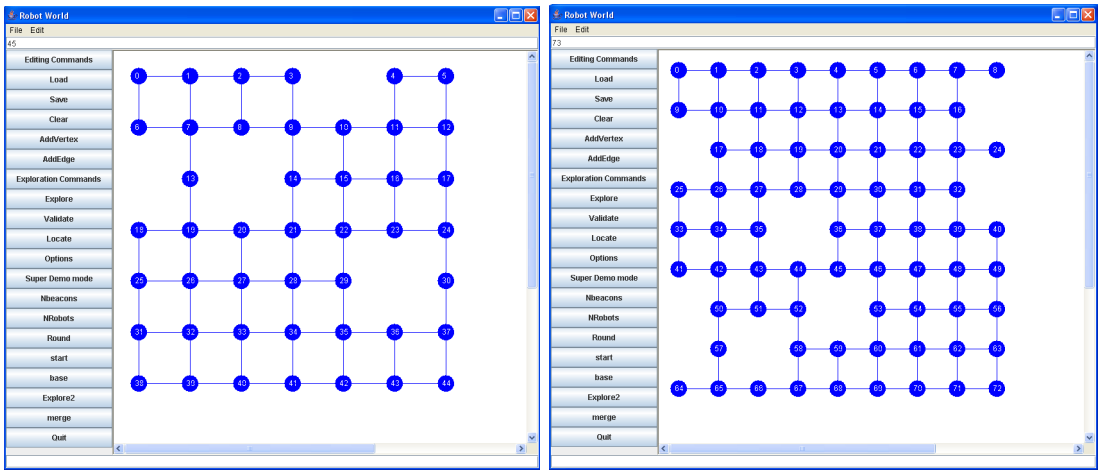
(a) Robot$_1$'s map (b) Robot$_2$'s map (c) Merged map

Exploration phase 5 (700×5 exploration steps) and merge phase 5
The full map is generated and the algorithm terminates



(d) Robot$_1$'s map (e) Robot$_2$'s map (f) Merged map

Figure 3.14: Multiple robot exploration on real system, phase 4–5.



(a) 7×7 lattice with 10% vertices removed (b) 9×9 lattice with 10% vertices removed

Figure 3.15: Lattice graph with holes.

Performance on lattice graphs

Finally we examine the performance of the algorithm on two-dimensional square lattices with holes, and compare the performance of the multiple robot exploration algorithm against the single robot exploration algorithm. Two-dimensional square lattices with holes represent an environment that is often encountered in the interior of modern buildings. Figure 3.15 shows two sample lattices with 10% of their vertices randomly removed. The performance of the algorithm is measured by $TaskCost$ defined in Equation 3.2.

The experiments were performed on lattices of various sizes. For each size lattice, measured by the number of vertices n in the lattice, the test was repeated 10 times, each using both the multiple (two robots) and the single robot exploration algorithm with the robot(s) starting at a randomly chosen starting place in the lattice of the same size n. For lattices (with holes) of the same size n, the number of holes are the same but the location of holes of the graph are generated randomly. (Due to the random distribution of holes,

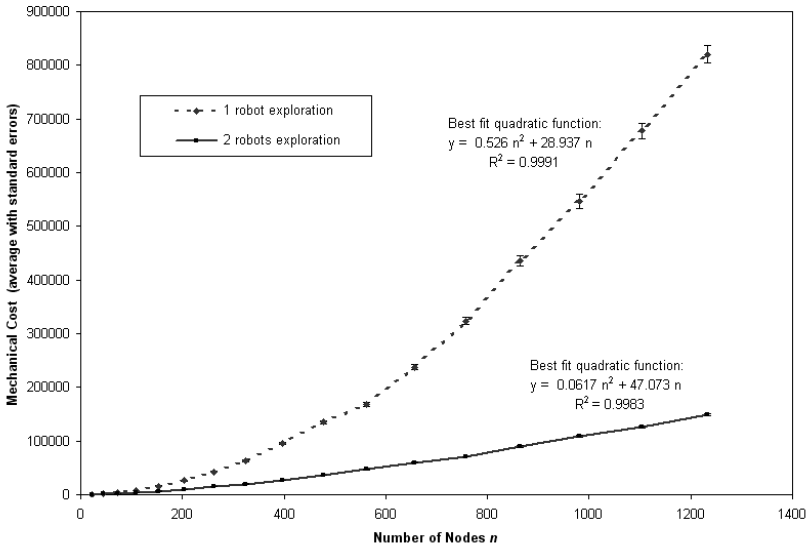


Figure 3.16: Single and multiple robot exploration on lattices with various sizes.

the number of edges of the same size graphs differs slightly from one to another.) The same random hole graphs are used in both the single and multiple robot explorations. Throughout the experiments, the rendezvous schedule is 100 mechanical steps and the starting point is the rendezvous place. The average $TaskCost$ of both the algorithms is shown in Figure 3.16, along with the standard errors. For both the single and multiple robot exploration, the best-fit quadratic functions of their average task cost over the lattice size n (number of vertices in the lattice) are generated as well. Figure 3.16 shows that even with the distribution of noise due to random effects, exploration with two robots consistently outperforms the exploration using single robot. For graphs of over 170 vertices, the multiple robot exploration presents more than half the reduction of the cost ($y_{2robots} \leq \frac{1}{2} y_{1robot}$ when $n \geq 170$), meaning that the exploration with two robots performs better than performing the single robot exploration twice. Moreover, when the graph size n is sufficiently large, the multiple robot exploration algorithm gives nearly 9

times improvement in performance over the single robot exploration algorithm, i.e., two robots complete the exploration task 9 times faster than one robot exploration.

3.5 Summary

This chapter first formally defined the world and robot models used. The same environmental representation as [14] was used but due to the deployment of multiple robots, the definitions of the robots' perception and operations are enriched and redefined, e.g., robots can sense each other, can communicate with each other. The model also made necessary synchronization assumptions, in which an atomic sequence of actions is defined, which simplifies synchronization of the robots. A detailed description of the multiple robot exploration algorithm was then presented. Fundamental issues involved in the merge process are presented in detail. In addition to the marker-based disambiguation technique that is similar to that in [14, 15], strategies such as 'electronic fusing' and 'direct fusing' were developed to reduce the mechanical work required by the robots. The chapter also presents a detailed formalism of one version of the possible merge algorithm, followed by proof of its correctness. To compare different solutions, a performance metric for general multiple robot exploration is developed, which accommodates both parallel and sequential work. Some sample graph explorations were presented along with a simple comparison of the multiple robot algorithm to the algorithm of Dudek et al. [14, 15]. The next chapter presents a number of potential improvements that can be made to both the original solution of Dudek et al. as well as to portions of the multiple robot exploration algorithm.

4 Enhancements to the basic algorithm

This chapter considers a number of potential improvements that can be made to both the original solution of the single robot exploration algorithm ([14, 15]) as well as to portions of the basic multiple robot exploration algorithm described in Chapter 3, e.g., the merging process and rendezvous scheduling. We begin by introducing two ideas that can be used to improve both the exploration process and the merging process in the multiple robot exploration algorithm. Enhancements to the exploration process can also be applied to the single robot exploration algorithm described in [14, 15]. The first enhancement is based on the use of neighbor information to disambiguate locations, the second enhancement is based on opportunistic communication when robots encounter each other 'accidentally' during exploration.

4.1 Exploiting neighbor topology information

In the original work of [14, 15] and the algorithm described earlier in this report the signature of the vertex v is its degree $d(v)$. Here we extend the signature to include its immediate neighbor topology. The neighbor topology information includes the degree of each neighbor and the label of the neighbor. Denote the extended signature of a vertex v as $sig(v)$. Then $sig(v)$ for a vertex v of degree k is an ordered set of tuples

$$sig(v) = \{(d_1, l_1), (d_2, l_2), ..., (d_k, l_k)\}$$

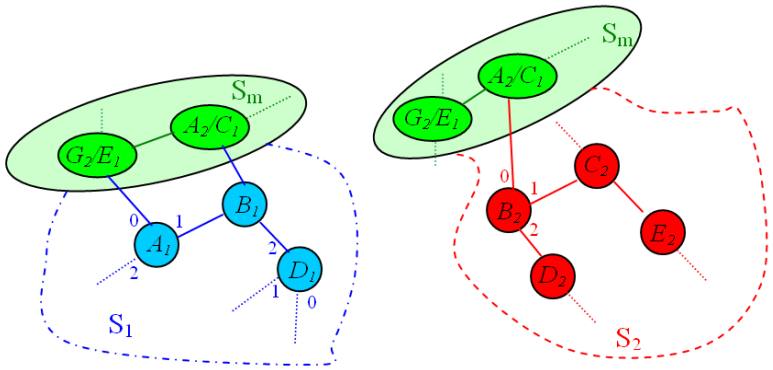


Figure 4.1: Retrieving neighbor information from maps.

where (d_i, l_i) represents (neighbor) vertex down edge i $(0 \leq i < k)$ of vertex v: d_i is the degree of vertex found down edge i and l_i is the label of the vertex down edge i. The label may be either *local* (this is a label assigned by the individual robot) or *global* (this is a label agreed to by the robots as a result of merging). If edge i is an unexplored edge, then d_i and l_i are both *null*. Note that $|sig(v)| = d(v)$. The order of the element in the signature set is based upon the normal enumeration of edges of v, but with an arbitrary initial orientation. For vertex B_2 in Figure 4.1, following the local edge ordering at B_2, the extended signature $sig(B_2) = \{(3, A_2/C_1), (3, C_2), (2, D_2)\}$. Note that C_2 and D_2 are local labels (known by robot$_2$) whereas A_2/C_1 represents a global (common) label.

4.1.1 Exploiting neighbor information in merging process

We first consider the exploitation of the extended signature (neighbor topology information) in the merge phase. We explore how to retrieve the extended neighbor information and how this neighbor information can help disambiguate potentially confusing vertices.

Aligning signatures Since the order of elements (neighbors) in a vertex signature follows the normal enumeration of edges at the vertex, in comparing the signatures of

two vertices we need to align possible corresponding exits (edges) and examine if they lead to the same vertex. In comparing two arbitrary vertices where no alignment information is available, we may need to take into consideration all permutations (e.g., all cyclic shifts). Due to the availability of some alignment information in the merging process, however, not all the permutations are 'valid' and need to be compared. This partial alignment information comes from the fact that for a vertex v_1 in S_1 to be a potential counterpart vertex of a frontier edge's uncommon (exterior) end vertex v_2 in S_2, *the edge of* v_1 *that corresponds to the frontier edge must be an unexplored edge.* For a vertex v_1 having n unexplored edges, there are at most n permutations that are 'valid' and need to be compared. Suppose we need to compare the signatures of vertex D_1 (having 2 unexplored edges) in S_1 with the exterior vertex B_2 in S_2 (Figure 4.1), there are two valid permutations in their signatures, i.e., permutation $\{(3, A_2/C_1), (3, C_2), (2, D_2)\}$ vs. $\{(null, null), (null, null), (3, B_1)\}$, and permutation $\{(3, A_2/C_1), (3, C_2), (2, D_2)\}$ vs. $\{(null, null), (3, B_1), (null, null)\}$. On the other hand, comparing the signature of vertex A_1 (having one unexplored edges) with that of B_2 involves only one valid permutation, i.e., $\{(3, A_2/C_1), (3, C_2), (2, D_2)\}$ vs. $\{(null, null), (3, G_2/E_1), (3, B_1)\}$. Each aligned signature permutation establishes pairs of neighbor that are 'pointed to' by the aligned exits in the permutation, e.g., $(3, C_2)$ vs. $(3, G_2/E_1)$. We call such two neighbors that are pointed to by the aligned exits in the permutation as a *neighbor pair*. Note that in the above neighbor pair, $(3, C_2)$ cannot match $(3, G_2/E_1)$ as the label G_2/E_1 is known commonly (globally) and thus both robot$_1$ and robot$_2$ agree to its labelling, whereas label C_2 is (locally) known by robot$_2$ only.

Comparing signatures – radius 1 Given two extended signatures of vertex v_1 (in S_1) and v_2 (in S_2), they are compatible if the vertices have the same degree ($|sig(v_1)| = |sig(v_2)|$) and, among the valid permutation(s), there exists at least one permutation

such that all of the neighbor pairs in the permutation are compatible. A neighbor pair represented by $(d_i, l_i) \in sig(v_1)$ and $(d_j, l_j) \in sig(v_2)$ is considered compatible if at least one of the following three properties holds:

(i) One or both of the two neighbors are unknown.

$d_i = null$ or $d_j = null$

(ii) Both of the neighbors are commonly known and they have the same common label.

$d_i \neq null$ and $d_j \neq null$, l_i and l_j are global labels, $l_i = l_j$

(iii) Both of the neighbors are locally known and have equal degree.

$d_i \neq null$ and $d_j \neq null$, l_i and l_j are local labels, $d_i = d_j$

Comparing signatures – radius n In the previous approach the signature was extended by considering local signatures one edge traversal from the vertex being considered. The signatures were compatible if there was some permutation of the orderings of the edges such that the information known about the vertices were consistent. We can trivially extend the approach to any radius 'r'. In the radius 1 algorithm, step (iii) of the consistency check can be replaced with a recursive call to the consistency check. Note that this recursion must have some maximum limit in order to bound the radius of the search.

Correctness of the enhancement Incorporating this enhancement into the merge process of the basic multiple robot exploration algorithm does not violate the correctness of the algorithm. The extended signature (neighbor information) 'filters out' (disambiguates) some vertices that would have to be visited in the basic algorithm. But any vertex that is filtered out cannot be a valid match, due to the fact that the signature comparison process only disambiguates vertices for which there is no possible permutation of

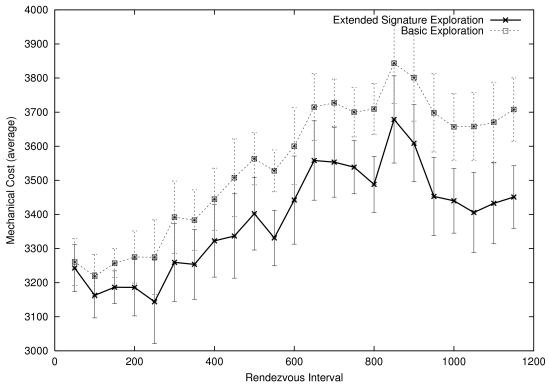
edges that allows a valid matching between the two subgraphs.

Evaluation and performance Using extended signatures in the merging process does not require coordination between robots, so as in the basic merge algorithm described in Chapter 3, there is one subphase in each merge phase. So the cost evaluation of the enhanced multiple robot exploration algorithm (basic multiple robot exploration algorithm with this enhancement incorporated) is the same as the cost evaluation for the basic algorithm, which is given by Equation 3.2.

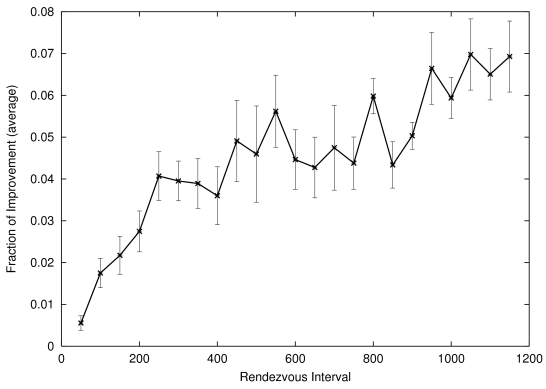
Incorporating the enhancement into the basic exploration algorithm does not require additional mechanical movements of the robots nor does it require any additional communication between them. All it does is reduce the number of potential vertices that must be visited in the basic algorithm when disambiguating locations. Moreover, using the extended signature does not change the order in which vertices are merged in the basic algorithm and therefore does not change the results of each merging and exploration process in the basic algorithm. So the operation of the enhanced algorithm under the same conditions as the basic algorithm (e.g., same graph, same starting place and orientation, same rendezvous scheduling and task division mechanism) will always produce better or equal performance.

To compare the performance of the enhancement against the basic algorithm, experiments were conducted using both the basic exploration algorithm and the enhanced exploration algorithm. In the enhanced exploration algorithm an extended signature of radius 2 is used in the merge phases. Experiments were performed on lattice graph with holes[†] (10×10 lattice graph with 20% of its vertices removed) using different rendezvous intervals. The test for each sample rendezvous interval was repeated 10 times, each with randomly generated holes in the graph and using both the original and extended signa-

[†]All lattice hole graphs used in this and the following experiments are lattice graphs defined on a torus (they are 'boundary-less').



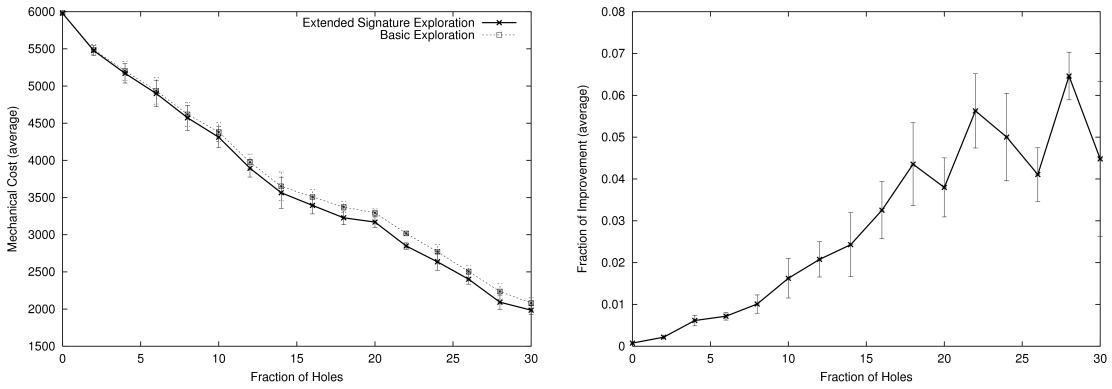
(a) Average mechanical cost



(b) Average improvement (fraction)

Figure 4.2: Using extended signatures in merging (varying rendezvous intervals).

tures exploration algorithm. Results are shown in Figure 4.2, where both the average cost of the algorithms and the average relative improvement of the enhanced algorithm over the basic algorithm are reported, along with the corresponding standard errors. First we see that the average cost for both algorithms increases as the rendezvous interval becomes longer. Results show that for all rendezvous intervals, the enhanced algorithm outperforms the basic multiple robot exploration algorithm, and that the improvement increases as the rendezvous interval becomes longer. To investigate the dependency of the extended signature's disambiguation ability on the graph topology, another set of experiments were conducted with graphs of varying homogeneity. In the experiments 10×10 lattice graphs with varying fraction of holes were explored using both the enhanced algorithm and the basic algorithm (with a fixed rendezvous interval). Again each condition was repeated 10 times and both the average cost and average relative improvement are reported, along with the standard errors (Figure 4.3). Results show that for all graphs, the enhanced algorithm produces positive improvements.



(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.3: Using extended signatures in merging process (varying homogeneity).

4.1.2 Exploiting neighbor information in the exploration process

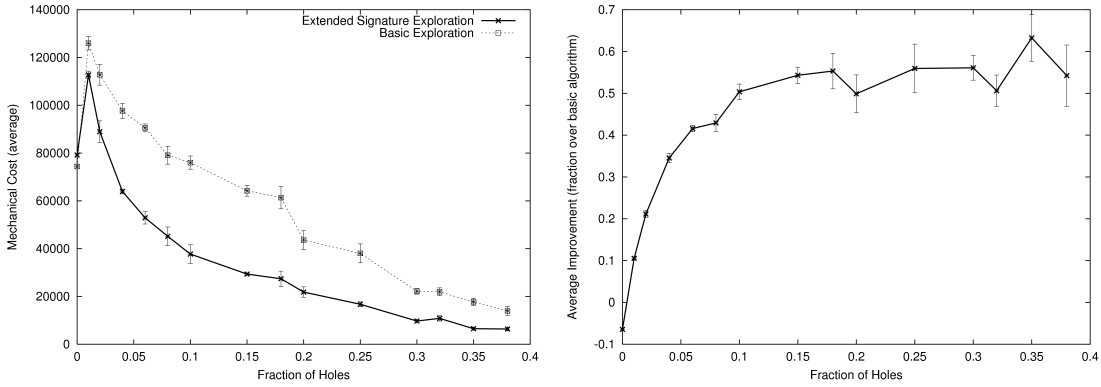
The extended signature can be used for disambiguation purposes in the exploration process as well. Since this enhancement does not require the cooperation of multiple robots, it can be used in both the single robot exploration case [14, 15] and in exploration phases in multiple robot exploration case. In the exploration process, the vertices to be disambiguated are the unknown place where the marker is dropped and the known places that are potentially confusing, i.e., known vertices that have unexplored edge(s) and have the same degree (local signature) as the unknown place. In comparing the signatures of the vertices, we also have alignment information for the signatures, due to the fact that for a potentially confusing vertex v to (potentially) be the counterpart vertex of the unknown place, then among v's incident edges, *it must be one of its unexplored edge(s) that corresponds to the edge incident on the unknown place which the robot used to enter this location.* Therefore, as in the merge phase, if v has n unexplored exits (edges), there are at most n permutations that are possible. Unlike the merge phase where signatures can

be retrieved from the partial maps of the individual robots, in exploration phase existing neighbor information for an unknown vertex is not 'rich' enough for efficient disambiguation. Under the basic exploration algorithm, for an unknown vertex where the marker is dropped, the only neighbor information available is about the vertex where the robot (marker) came from. Assume the robot travels from vertex v_{from} to an unknown vertex v_{new} to drop the marker. Also assume that the degree of v_{from} is 4 and the robot senses the degree 3 at v_{new}. Then the signature for v_{new} is $\{(4, v_{from}), (null, null), (null, null)\}$. Clearly the comparison of the signatures would always result in many compatible matches. To obtain useful neighbor information, extra mechanical steps are required. For example, when a robot drops a marker at vertex v_{new}, it could also explore all of the incident edges (except the edge by which it entered) and sense the degree there and use this information to construct a more powerful local signature. Note that even with these extra mechanical moves only the degree information can be retrieved. In the example, the signature of the unknown place will be enriched to $\{(4, v_{from}), (d_2, null), (d_3, null)\}$.

Correctness This enhancement is trivially correct as the extended signature is only used to filter out locations that can be rejected as they demonstrate that no subgraph match exists.

Evaluation and performance Since there is no additional coordination between robots, the exploration phase has only one subphase, which is the phase itself. So the evaluation mechanism of the enhanced algorithm is the same as the basic algorithm into which the enhancement is incorporated, i.e., when incorporated in single robot exploration algorithm [14, 15], the cost of the enhanced algorithm is the mechanical cost of the single working robot, when incorporated into the exploration phases of the basic multiple robot exploration algorithm, the cost of the enhanced algorithm is given in Equation 3.2.

The enhancement does not change the order in which new places are explored, but it

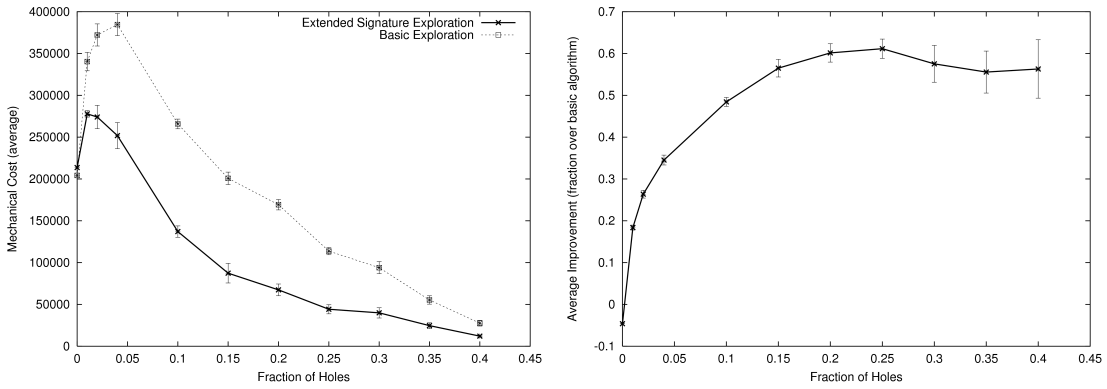


(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.4: Using extended signature in exploration phase (20×20 lattice).

does require extra mechanical cost to retrieve the extended signatures. Experiments were performed to compare the performance of the single robot exploration algorithm with the enhancement incorporated against the performance of the basic single robot exploration algorithm [14, 15]. Experiments were conducted on both 20×20 and 28×28 lattices with varying number of holes (0%–40%). Each condition was repeated 10 times using both the algorithms. The results are shown in Figure 4.4–4.5, where both the average cost of the algorithms and the average relative improvement of the enhanced algorithm over the basic algorithm are reported, along with corresponding standard errors. We can see that in both graphs, when there are zero holes missing, i.e., when the graph is completely homogeneous, the performance of the enhanced exploration is worse than the basic algorithm, which is trivially true because extra mechanical costs have been spent at each new place but the disambiguation tasks cannot benefit from the extended signatures. Even with 1% holes, the enhanced algorithm shows improved performance. In this example cost reductions of up to 60% are achieved.



(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.5: Using extended signature in exploration phase (28×28 lattice).

4.2 Exploiting communication information

During independent exploration robots may encounter each other unintentionally. The basic model described in Chapter 3 allows a robot to sense the presence of the others and to communicate but the communication between robots only occurs at specific, pre-arranged rendezvous locations. As the robots move about they may encounter other robots. How can opportunistic communication be exploited by the robots? Here communication during both the exploration phase and the subsequent merge phase of the multiple robot exploration algorithm is explored. There are many potential advantages to communicating with other robots when they are encountered during the exploration algorithm. Only two potential situations are considered here: communication that later will aid in the merging process and communication that can be exploited immediately under certain conditions.

4.2.1 Exploiting communication information later in the merge process

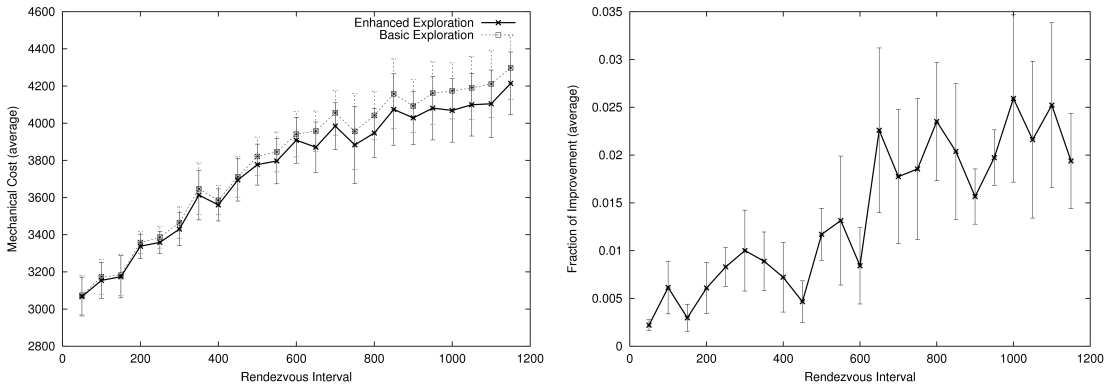
One useful piece of information that the robots can communicate when they encounter each other in the world is where they currently are in their own maps. For example, robot$_1$ may claim that this is (exterior) vertex v_{E1} in its independently explored map S_1 and robot$_2$ claims this is (exterior) vertex v_{E2} in its independently explored map S_2 (so v_{E1} and v_{E2} are not commonly known). Then this physical encountering implies that v_{E1} and v_{E2} refer to the same location in real world, i.e., v_{E1} and v_{E2} are counterpart vertices. Call this information the *partial matching* information between v_{E1} and v_{E2}. This is partial matching information in that unlike other commonly known vertex pairs in S_m the edge index correspondence between v_{E1} and v_{E2} is not known. This information, however, can be exploited in the subsequent merge phase. Suppose during the subsequent merge phase, base robot$_1$ selects a frontier edge that is incident on a common (interior) vertex $v_{I2/I1}$ in S_m and an uncommon (exterior) end v_{E2} in S_2. Before moving to the uncommon end v_{E2} to drop the marker, the partial matching record is examined. If the exterior end v_{E2} in S_2 partially matches some vertex v_{E1} in S_1, then the robot 'knows' that v_{E1} and v_{E2} refer to the same place in the real world. In this case the exterior end is not new to the base map and only a new edge ($v_{I2/I1}$, v_{E1}) needs to be added to the base map. Since now v_{E1} and v_{E2} are counterpart vertices, instead of dropping the marker at v_{E2} and searching for the marker, the robot can start establishing the edge ordering of the new edge at v_{E1} (back-link validation) immediately by dropping the marker at the common (interior) end $v_{I2/I1}$, and traveling to v_{E1} in S_1, where it traverses unexplored edges on v_{E1} to perform the back-link validation. Note that when two robots meet in a vertex, one or both of them may be in an 'unknown' place in its map, e.g., two robots encounter when one or more of them is coming to drop its marker at a new place. In such case, the encountering information is recorded and when the unknown place is disambiguated by the exploration therefore becomes known, the record is updated.

Correctness Since robot$_1$ and robot$_2$ have physically met, v_{E1} and v_{E2} refer to the same place in the real world, i.e., v_{E1} and v_{E2} are counterpart vertices. Therefore during the merge phase, if the robot drops its marker at v_{E2} in S_2, it must find it when it visits v_{E1} in S_1. The process of establishing this property during the subsequent merge phase can thus be avoided. These steps include dropping the marker (at v_{E2}), searching the marker, finding the marker (at v_{E1}) and picking up the marker, and coming back to common (interior) place $v_{I2/I1}$ (to drop marker).

Evaluation and performance Although robots meet and communicate, there is no extra coordination and synchronization required, so the exploration phase still has one subphase, which is the whole phase. Using this communication information in the following merge phase does not change the evaluation process for the merge phase either. So the cost evaluation mechanism for the enhanced multiple robot exploration algorithm is the same as that for the basic multiple robot exploration algorithm (Equation 3.2).

Incorporating the enhancement into the basic multiple robot exploration algorithm does not introduce any extra mechanical cost. It also does not affect the portion of the basic algorithm that defines the order in which the vertices are explored in the exploration phase, nor does it affect the order in which vertices are merged in the merging process. So it does not change the result of each exploration and merging process. All it does is reduce the potential searching steps required by the base robot when disambiguating locations in the merging process. So for the operation of algorithms with exactly the same condition (e.g., same graph, same starting place and orientation, same rendezvous scheduling) the performance of the enhanced exploration algorithm will be better or equal to the basic multiple robot exploration algorithm.

Experiments were conducted using both the basic exploration algorithm and the enhanced exploration algorithm. Experiments were first performed on a 10×10 lattice graph

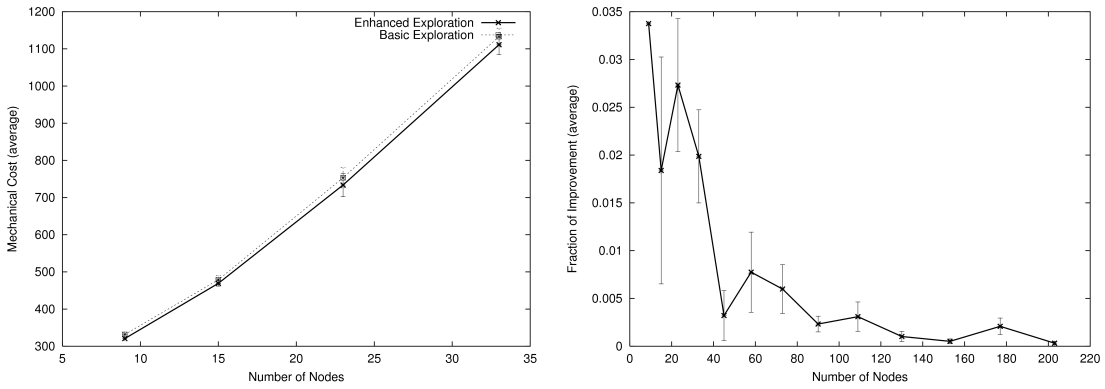


(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.6: Exploiting communication information in merge phase (varying rendezvous interval).

with 20% of its vertices removed, using different rendezvous intervals. Each condition was repeated 10 times using both the basic and the enhanced algorithm. The average cost and relative improvement along with standard errors are reported in Figure 4.6. We can see that for all rendezvous intervals, the enhanced algorithm performs at least as well as the basic multiple robot exploration algorithm. To further examine the effects of the opportunistic encountering on the performance, further experiments were conducted on lattice hole graphs of varying sizes. Each condition was repeated 10 times using both the basic and the enhanced algorithm. The average cost and relative improvement are reported in Figure 4.7 along with standard errors. We can see from the results that the enhanced algorithm produces positive improvements. Moreover, as the graph size increases, the improvement decreases, due to the fact that the chance of robots encountering each other becomes less likely in larger graphs.



(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.7: Exploiting communication information in merge phase (varying graph size).

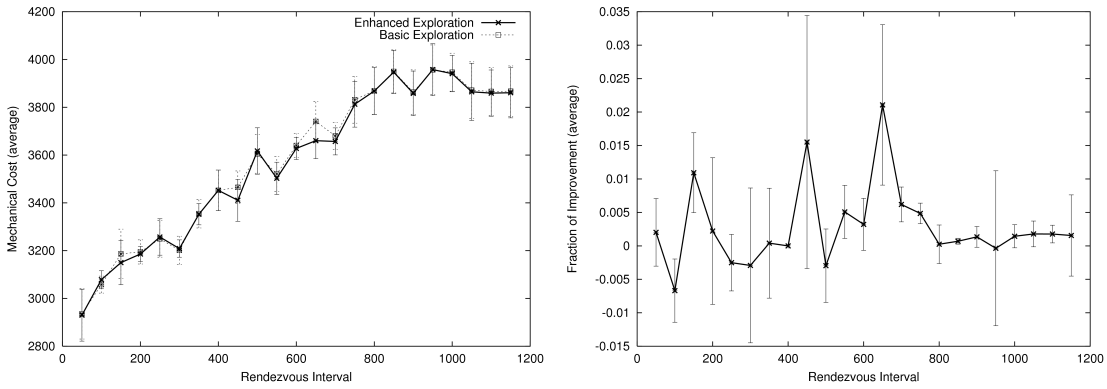
4.2.2 Exploiting communication information in the exploration phase

Accidental meeting of the robots can also lead to opportunistic information that can be exploited (immediately) during exploration. Consider the following scenario during independent exploration. Robot$_1$ follows one unexplored edge of its vertex v_{E1} to an unknown place (to drop its marker) where it encounters robot$_2$ and they communicate. Robot$_2$ 'tells' robot$_1$ that this place is the commonly known place globally labeled vertex $v_{I2/I1}$ in both maps (as a result of previous merging). Then robot$_1$ knows that this unknown place is actually the common vertex $v_{I2/I1}$ in its map, so this 'unknown' place is not new to its map and only a new edge (v_{E1}, $v_{I2/I1}$) should be added to its map. Robot$_1$ can now start establishing the edge ordering of the new edge at $v_{I2/I1}$. If robot$_1$ drops its marker at the 'unknown' place as planned, it will find it at $v_{I2/I1}$ in its map. So instead of dropping the marker at the 'unknown' place and searching the marker, robot$_1$ can start the back-link validation process, i.e., go back to v_{E1} and drop its marker there. Then go to $v_{I2/I1}$ in its map and explore all unexplored edges until it senses the marker at v_{E1}.

Correctness Similar to the above enhancement in the merge phase, the physical encountering of the two robots at a commonly known place $v_{I2/I1}$ means that the 'unknown' place is actually $v_{I2/I1}$ in robot$_1$'s map. So if the marker is dropped at the 'unknown' place, it will be found at $v_{I2/I1}$. The process of establishing this property during the current exploration phase can thus be avoided. This includes the exploration steps of dropping the marker (at the 'unknown' place), searching for the marker, finding the marker (at $v_{I2/I1}$) and picking it up.

Evaluation and performance Incorporated into the basic algorithm, this enhancement reduces the potential steps required in the basic algorithm in exploring new locations during exploration. In general, this potential reduction of steps in the exploration process is expected to produce a reduction of the overall task cost by enabling robots to explore more within each pre-defined rendezvous interval. However, in contrast with the previous enhancement where each reduction (by the base robot in the merge phase) contributes to the overall cost reduction, the reduction of steps due to this enhancement does not necessarily lead to an overall cost reduction. Assuming complete parallel work in the exploration phase, the cost of each exploration phase is bounded by the cost of the 'busier' robot in that phase. So the reduction of an individual robot's cost may not necessarily affect the overall cost. For example, one robot may benefit from the enhancement so as to finish its exploration with fewer steps than needed in the basic algorithm but it may have to wait at the rendezvous place for the other robot to finish, or, the earlier robot may be able to explore more than in the basic algorithm but produces work that duplicates that of the other robot. Moreover, by changing the exploration result of an exploration process, the enhancement might change the order in which vertices are merged and explored in subsequent processes, resulting in different performance.

Similar to the earlier enhancement, experiments were conducted both on a lattice hole



(a) Average mechanical cost

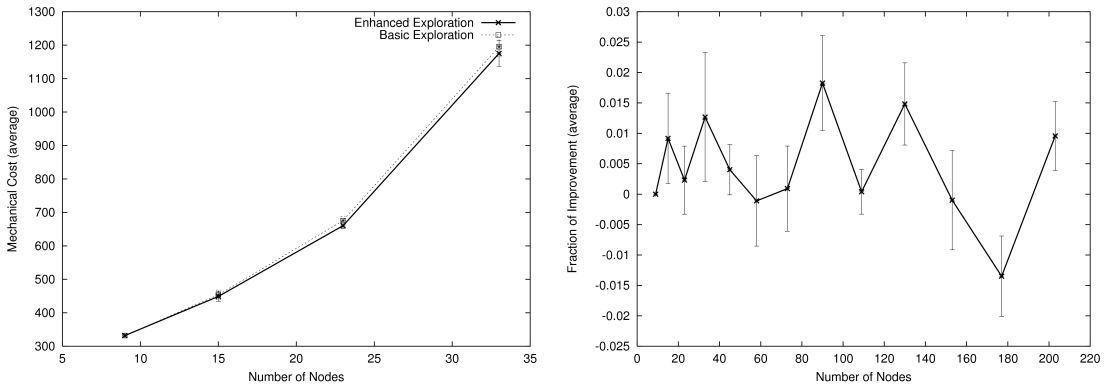
(b) Average improvement (fraction)

Figure 4.8: Exploiting communication in exploration phase (varying rendezvous interval).

graph using different rendezvous intervals, and with a fixed interval on graphs of varying sizes. Results are shown in Figures 4.8–4.9. First we can see that the two algorithms produce similar performance. Results show that this enhancement can introduce both positive and negative improvement over the basic algorithm, and in the cases explored here very limited effect was found.

4.3 Using multiple robots in the merge phase

In the merge phase we have two robots and we can exploit the existence of both robots and both markers in several ways. One such potential exploitation occurs in the back-link validation stage after the marker is found. Recall that in the basic merge algorithm, when a marker that is dropped from vertex $v_{I2/I1}$ (the interior vertex of the frontier edge) is found at vertex v_{E1} in the base map, the robot needs to establish edge ordering of the new edge ($v_{I2/I1}$, v_{E1}) at v_{E1} using its sole marker. Establishing such a local edge ordering of the new edge requires that the base robot picks up the marker and transits to $v_{I2/I1}$, drops the marker at $v_{I2/I1}$ and goes back to v_{E1}. At v_{E1} the robot tries to go



(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.9: Exploiting communication in exploration phase (varying graph size).

out of each unexplored edge incident on v_{E1}, until it senses the marker. To facilitate the validation task, we can let another robot (robot_2) act as a second marker to reduce the effort required to merge the maps. For example, robot_2 could move with robot_1 (in the basic merge algorithm robot_2 is immobile during the merging process). When robot_1 moves to disambiguate a node, robot_2 can remain at the other end of the edge. Then when back-link validation is being performed the second robot can be used to generate a unique signature of the vertex at the other end of the link.

Note that significant communication and synchronization efforts are needed to realize cooperation between the robots. Initially, the robots are at the same vertex (the rendezvous place) after their independent explorations. When the base robot chooses a common (interior) vertex $v_{I2/I1}$ that has frontier edge(s) incident on, it communicates with robot_2 so that robot_2 can also move to $v_{I2/I1}$. To ensure the presence of robot_2 at $v_{I2/I1}$ when back-link validation is needed, robot_1 must meet robot_2 again before it can start its search for the marker. Robot_2's presence at $v_{I2/I1}$ can now be used to provide a unique signature at $v_{I2/I1}$ and therefore can be used to avoid much of the back-link

validation effort (mechanical moves). We show below that the need for both robot$_1$ and robot$_2$ to meet to resynchronize is 'free' in terms of the algorithmic cost as robot$_1$ always moves back to $v_{I2/I1}$ after dropping its marker, and will move back to $v_{I2/I1}$ (again) in marker-searching stages of the merge algorithm.

Correctness Robot$_1$ will not start its searching motion away from $v_{I2/I1}$ until robot$_2$ is also present at $v_{I2/I1}$. This ensures the presence of robot$_2$ at the other end of the link for back-link validation. The enhancement replaces the back-link validation process in the basic merge algorithm, the rest of the steps are the same as that in the basic merge algorithm.

Evaluation and Performance Since two robots have to meet again in $v_{I2/I1}$ before robot$_1$ starts its search, there are now both parallel and sequential work in the merge phase. Again we assume that robots can operate in complete parallel (this can be the case in real life situation). As described above in order to determine the mechanical complexity of the enhanced algorithm the merge phase of the algorithm must now be divided into subphases so that work in each subphase is completely parallel. Since this process repeats for each round of marker-based disambiguation, we divide each iteration of the merge phase into two subphases. In subphase$_1$ robot$_1$ selects a common vertex $v_{I2/I1}$ that has frontier edge(s) and informs robot$_2$. Then robot$_1$ goes to v_1, traverses a frontier edge at v_1, drops its marker at the other (exterior) end and comes back to v_1. At the same time the robot$_2$ transits to $v_{I2/I1}$ and remains there. Subphase$_1$ ends when the two robots meet in $v_{I2/I1}$. Subphase$_2$ is the search and validation phase by robot$_1$ and ends when robot$_1$ finishes its work and moves back and meets robot$_2$ again at $v_{I2/I1}$. This is shown in Figure 4.10. Suppose the merge phase M_i has l iterations. Then the number of subphases in M_i is $2l$. The enhancement algorithm can be evaluated using the task cost defined for the generic multiple robot exploration algorithm (Equation 3.1), where

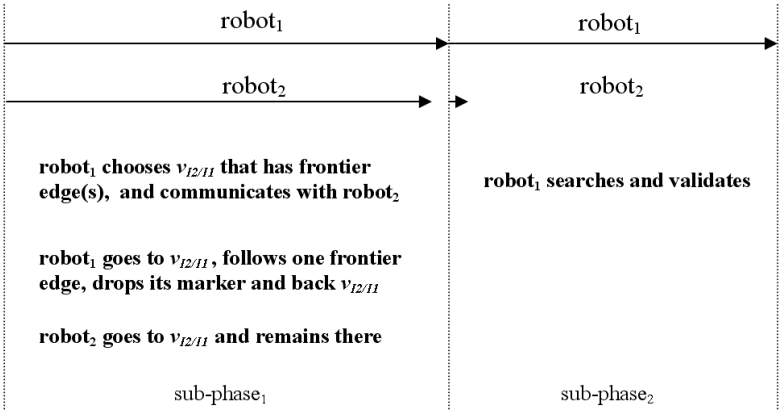


Figure 4.10: An iteration of the merge phase divided into subphases.

the cost for exploration phase E_i has $m = 1$ (one subphase), cost for merge phase M_i has $m = 2l$.

While the enhancement does not change the order in which vertices are merged and explored in the basic algorithm, the comparison of the performance of the enhanced algorithm against the basic algorithm entails detailed analysis of the 'behavior' of the robots in both algorithms, due to the extra synchronization efforts involved in the enhancement. In subphase$_1$, both robots start from the same place and come to the same place ($v_{I2/I1}$). They can follow the same shortest path to $v_{I2/I1}$, so if the path length (number of edges) is d, then in this subphase robot$_2$ makes d mechanical moves whereas robot$_1$ (the base robot) makes $d + 2$ mechanical moves. (The two more mechanical steps are spent on the traversal of the frontier edge incident on $v_{I2/I1}$ to drop its marker and move back.) Assuming completely parallel work of the robots, the cost for the parallel work in subphase$_1$ (the maximum cost of the robots) is bounded by the cost associated with the base robot. The cost of subphase$_2$ is simply the cost associated with the base robot. For comparison purposes, we divide each iteration of the merge phase in the basic multiple robot exploration algorithm (without the enhancement) into two subphases according to the mechanical movement of the base robot. As for the enhancement, subphase$_2$ is

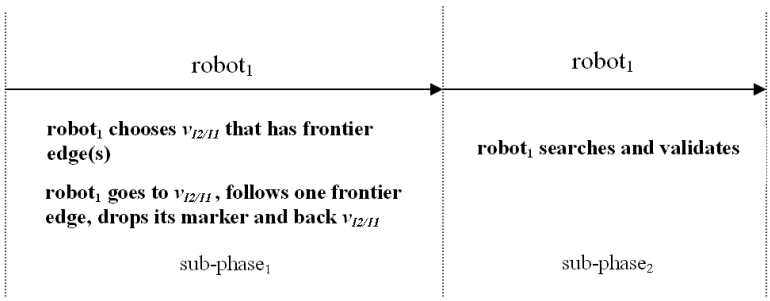
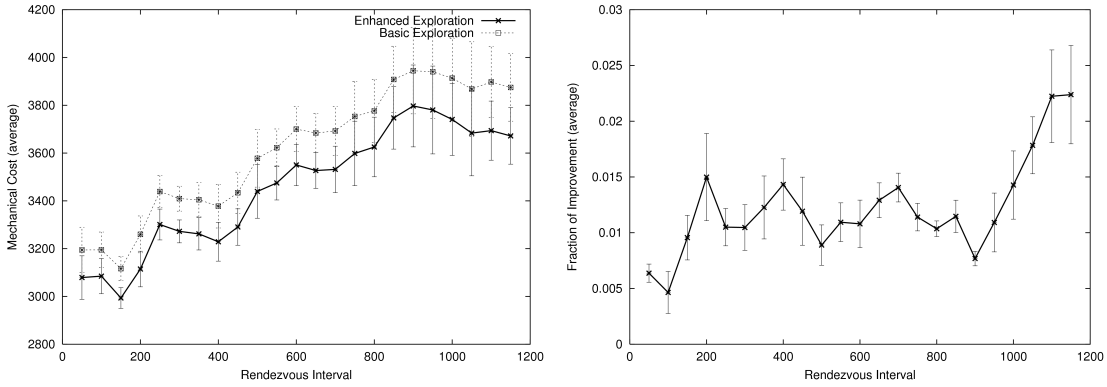


Figure 4.11: An iteration of basic merge phase divided into sub-phases.

search and validation phase, and the work before the search and validation belongs to subphase$_1$. Specifically, subphase$_1$ starts when the base robot chooses a place $v_{I2/I1}$ having frontier edge(s), and ends when the base robot finishes dropping its marker and moves back to $v_{I2/I1}$. Subphase$_2$ starts as the base robot starts its search and ends when the base robot finishes validating the unknown place (Figure 4.11). Since the enhancement does not change the order in which vertices are selected for merging, in subphase$_1$ of the basic merge algorithm the base robot also makes $d + 2$ mechanical moves, i.e., in terms of mechanical moves, robot$_1$'s behavior in subphase$_1$ of the enhancement is the same as that in subphase$_1$ of the basic algorithm, and the task cost of subphase$_1$ of the two algorithm is the same. In subphase$_2$, the marker is either found or not. Designed to facilitate back-link validation when the marker is found, the enhancement uses exactly the same process as in basic algorithm if the marker is not found. So when the marker is not found, the cost of subphase$_2$ is same as in basic algorithm. When the marker is found, the base robot will have made fewer validation traversals, and there are no other additional costs. So, in subphase$_2$, the cost for the enhancement algorithm is equal to or lower than the cost of subphase$_2$ of the basic algorithm. So the cost for each merge phase (sum of the two subphases) will be equal or lower than that in the basic algorithm. (As mentioned, the need for the synchronization is 'free' in terms of the algorithmic cost.) Exploration phases are not affected, therefore the task cost of the enhanced multiple robot exploration



(a) Average mechanical cost

(b) Average improvement (fraction)

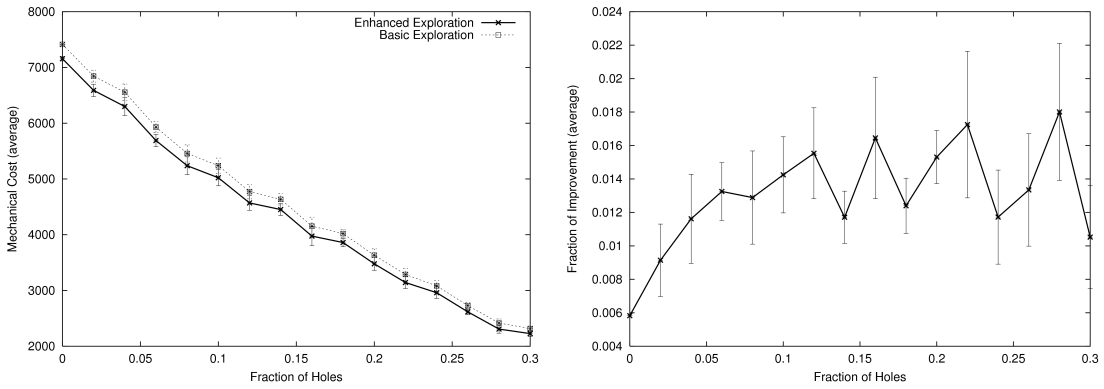
Figure 4.12: Using multiple robots in the merge phase (varying rendezvous interval).

algorithm is equal to or lower than the basic multiple robot exploration algorithm.

Experiments were conducted using both the enhanced algorithm and the basic multiple robot exploration algorithm. Similar to that for previous enhancements, one set of the experiments operates on the lattice with 20% holes using varying rendezvous intervals, another set of experiments operates on lattices of varying number of holes with fixed rendezvous interval. Each condition was repeated 10 times and results (average cost and fraction of improvement) are shown in Figure 4.12–4.13, along with standard errors. We can see that for all the experiments, the enhanced algorithm outperforms the basic algorithm. The results also show that the enhancement is not very sensitive to the homogeneity of the graphs.

4.4 Breadth-first exploration

This section and the next present two enhancements that address the exploration process. Since these two enhancements do not deal with the coordination of multiple robots, they can be used in both the single robot exploration case and the exploration phase of the



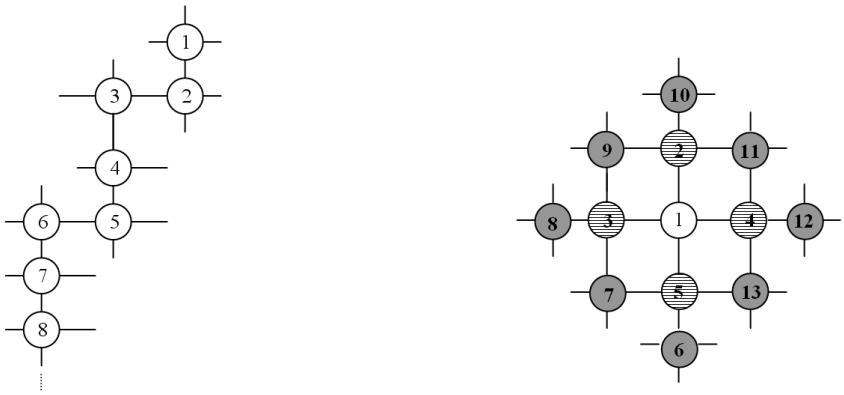
(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.13: Using multiple robots in the merge phase (varying homogeneity).

multiple robot exploration.

In the basic exploration algorithm, the robot always chooses the closest place to explore. (It uses a greedy heuristic based on depth-first search to define its route.) Consider the example in Figure 4.14(a) where a lattice graph is explored. Suppose the robot starts at vertex 1 and chooses the unexplored edge leading to vertex 2. According to the basic algorithm, it will be at vertex 2 when it finishes exploring vertex 2. Then it chooses an unexplored edge on vertex 2 leading to vertex 3. When finished with processing vertex 3, it chooses the unexplored edge at vertex 3 leading to vertex 4, and so on. The labels in Figure 4.14(a) illustrate the order of exploring new vertices in the basic algorithm. In this example, in disambiguating the unknown place later labelled vertex 4, vertices 1, 2 and 3 are potentially confusing vertices (they have unexplored edge(s) with the same degree as the new vertex). As a consequence, the robot has to go back to visit vertex 3, vertex 2, and all the way back to the furthest vertex 1. In this example, as the exploration proceeds, the depth-first selection of the closest new place to explore generates repeated traversals to the same vertices. To avoid the repeated



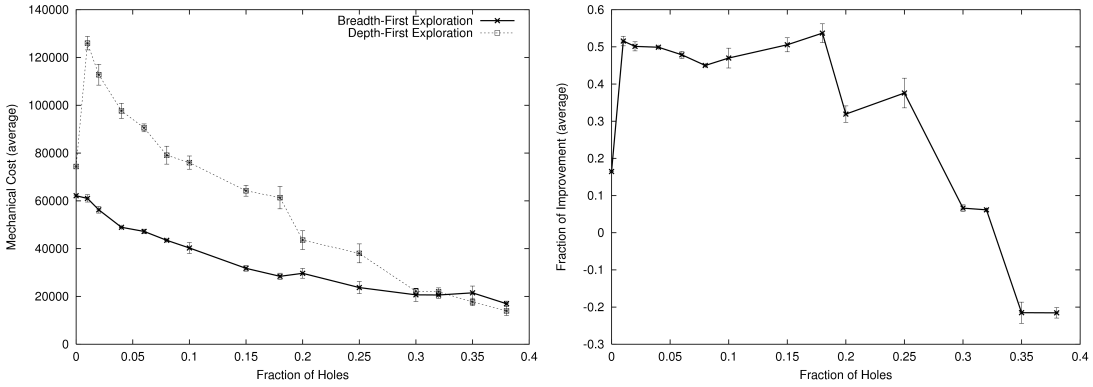
(a) Depth-first exploration on lattice

(b) Breadth-first exploration on lattice

Figure 4.14: Two different exploration strategies.

traversals, a natural alternative to the depth-first exploration would be to use breadth-first exploration, i.e., explore all unknown edges of vertex 1 first, then explore all unexplored edges of neighbors of vertex 1, and so on, as shown in Figure 4.14(b), where the labels and colors of the vertices illustrate the order and 'breadth' of the exploration. Using this approach, the traversal of repeated search for the same vertices in this example can be reduced. Note that this is not true in general as there exist examples for which a depth-first exploration is more efficient than breadth-first exploration. Empirically, however, it appears to be more efficient as will be demonstrated for lattice graphs.

Correctness This enhancement adopts a different way of exploring new places, the only difference with the basic algorithm is the order in which the unknown places are selected. The details of exploring each unknown place is the same as the basic algorithm, therefore incorporated in either the single robot exploration algorithm [14, 15] or the exploration phases of the basic multiple robot exploration algorithm described in Chapter 3, the correctness of the algorithms is not violated.



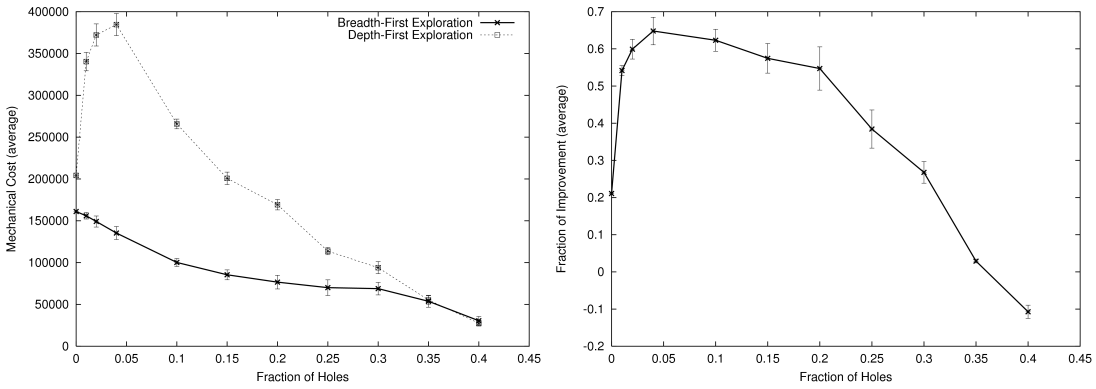
(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.15: Breadth-first exploration (20×20 lattice with holes).

Evaluation and performance Since there is no extra coordination between robots, the exploration phase has only one subphase, which is the phase itself. So when incorporated in either the single robot exploration algorithm or the exploration phases of the basic multiple robot exploration algorithm, the evaluation mechanism is the same as in the corresponding basic algorithms.

Experiments were conducted to examine the performance of the breadth-first exploration in the above example. The setting of the experiments is the same as the experiments conducted for the extended signature exploration, i.e., a single robot exploring on 20×20 and 28×28 lattice with varying number of holes. In contrast to the extended signature enhancement, the breadth-first exploration is expected to work well in less heterogeneous lattice graphs, where repeated traversals for disambiguation are more likely. As before, each condition was repeated 10 times and the average cost and relative improvement are reported in Figures 4.15–4.16. We can see that in both figures, the breadth-first exploration gives substantial improvement over the depth-first exploration approach for graphs with a low hole density. As the number of holes increases,



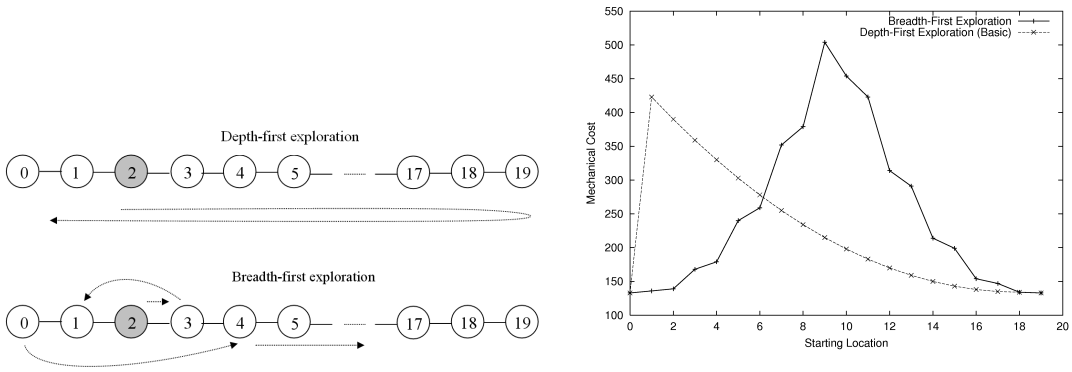
(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.16: Breadth-first exploration (28×28 lattice with holes).

the breadth-first exploration still provides an improvement but the improvement becomes smaller, and eventually when the graph is sufficiently heterogeneous (with more than 30% holes), the improvement vanishes.

To investigate other factors that might affect the performance of the breadth-first exploration, the operation of the algorithm on a string-like graph was examined. For a string with no loops, the depth-first exploration will explore in one direction until it reaches the end vertex of the string (hit the boundary) and then go back to explore the rest of the string in opposite direction, whereas with breadth-first exploration, the robot will keep on 'oscillating' between the two ends of the current known string, until one of the ends is the end vertex of the string (hit the boundary), and then go back to explore the rest of the string. An example of exploring a 20 node string is illustrated in Figure 4.17(a), where node 2 is the starting place and it is assumed that for a vertex with two unexplored edges, the robot always chooses the right-hand side edge to explore. Since after reaching one end vertex of the string (hitting the boundary) the robot's currently known string is fully explored, the rest of the exploration incurs no disambiguation traversals. Clearly



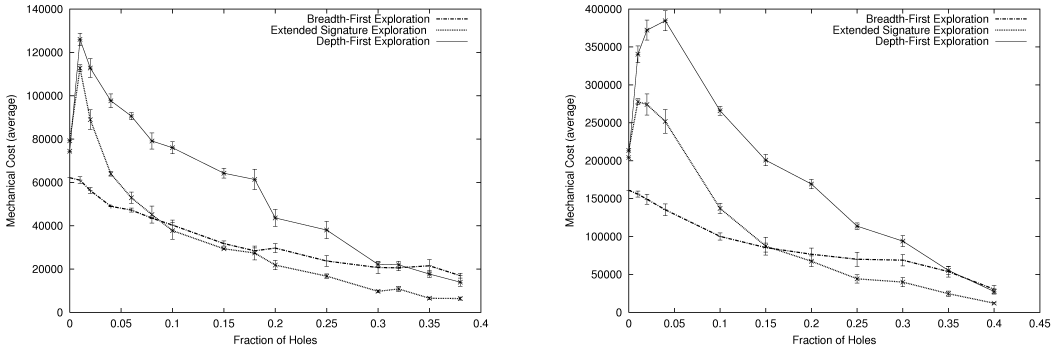
(a) Different exploration patterns

(b) Mechanical cost

Figure 4.17: Breadth-first exploration vs. Depth-first exploration on strings.

the starting place, which determines the distance to the end vertex (boundary), is one of the factors affecting the performance of both approaches. Experiments were conducted on exploring a 20 nodes string using both exploration strategies, with varying starting places. The results are shown in Figure 4.17(b). We can see that the best cases for both the strategies occur at the two ends of the string, because for both strategies the exploration is in one direction only and there are no disambiguation traversals. The worst case starting place for the two strategies are different. The worst case performance for the two strategies are different too.

It is interesting to compare the single robot breadth-first exploration, extended signature exploration and the basic (depth-first) exploration algorithm, as shown in Figure 4.18. When there are few holes, i.e., the graph is relatively homogeneous, the breadth-first exploration outperforms the extended signature algorithm. As the number of holes increases, the improvement from the breadth-first exploration decreases whereas the improvement from the extended signature increases, eventually when the graph is sufficiently



(a) On 20×20 lattice graph with holes

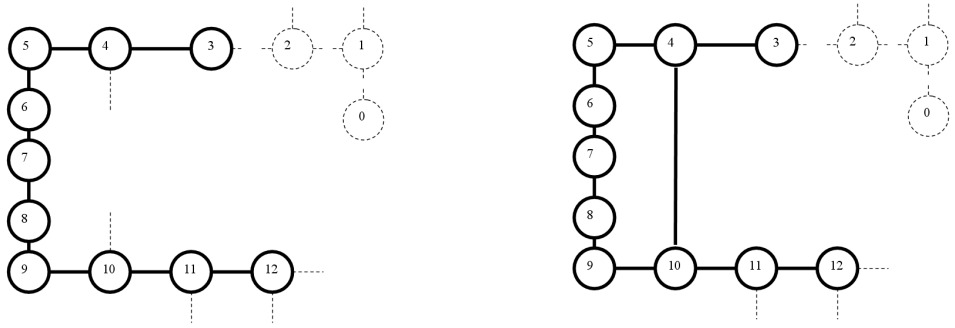
(b) On 28×28 lattice graph with holes

Figure 4.18: Three different exploration strategies.

heterogeneous, the extended signature exploration outperforms the breadth-first exploration.

4.5 Lazy exploration

Search is a critical but costly task, both in the exploration and merge phases. How 'hard' a search task is (the mechanical cost needed to visit all possible locations) depends on the number of vertices that need to be visited and where these vertices are located. In the single robot exploration algorithm [14, 15] a brute force strategy is adopted in prioritizing this search. Each incoming search task is performed, irrespective of how 'hard' it is (first come first served). Earlier we have seen that different exploration order may generate different task costs. This is because in a connected bidirectional graph a new vertex or edge can be approached and eventually validated via different routes with different search costs. One illustrative situation is shown in Figure 4.19. As the known subgraph grows (Figure 4.19(b) vs. Figure 4.19(a)), validating the unknown edge incident on vertex 3



(a) Validating edge (3,2) requires long path

(b) Validating edge (3,2) enjoys shorter path

Figure 4.19: Growth of known graph produces different search tour length.

(leading to vertex 2) requires a reduced search cost, i.e., the search tour to potentially confusing vertices (vertex 10, vertex 11, vertex 12) is reduced due to the newly added 'shortcut' edge (4,10).

We have developed a breadth-first exploration algorithm, which explicitly changes the exploration order adopted in Dudek et al.'s basic exploration algorithm. We present here another strategy that 'implicitly' changes the order in which new places are explored. This strategy is called 'lazy exploration'. In the lazy approach, the order of choosing new places is the same as that in single robot exploration algorithm (depth-first), but search tasks are prioritized, according to the 'difficulty' of the tasks. In the lazy exploration approach, a 'difficult' search task is put off to later steps. Here the difficulty of the search task for an unknown place is evaluated based on the length of a search tour visiting all potentially confusing vertices of the unknown place, measured in terms of the mechanical cost (the number of edge traversals) required. We develop two techniques to decide the acceptance or delay of a currently selected search task based on its difficulty. The first technique is a deterministic approach, in which the decision is made based on the direct comparison of the search length and a pre-defined threshold, i.e., a task that requires a search length

that is below the threshold is accepted and the task is rejected otherwise. The second technique is a probabilistic approach, in which the decision is made probabilistically based on the length of the search tour.

Correctness Similar to the breadth-first exploration, in the lazy exploration new portions of the world are explored in a different order from the basic algorithm. And we show below all of the tasks will be accepted eventually. The details of exploring each unknown place is the same as the basic algorithm. Therefore incorporating this technique in either the single robot exploration algorithm [14, 15] or in the exploration phases of the basic multiple robot exploration algorithm does not violate the correctness of the algorithms.

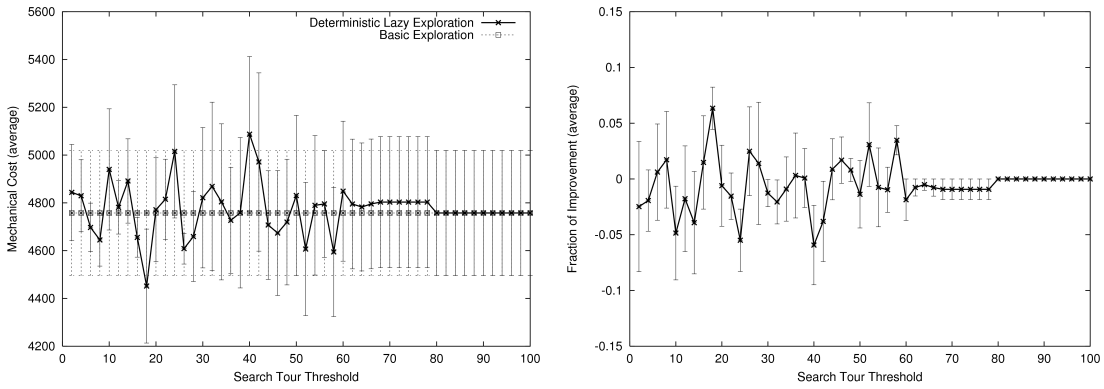
Evaluation and performance Since no additional coordination is required between the robots, as in the basic algorithms the exploration phase has only one subphase, which is the phase itself. So the evaluation mechanism is the same as that for the basic algorithms.

Experiments were conducted to evaluate the performance of both the threshold based deterministic and the probabilistic lazy exploration approaches. In the deterministic approach once a task is 'delayed', the robot tries other tasks and the delayed task will not be considered until some other task is accepted and processed. Once some other task is performed, all delayed tasks become 'normal' tasks and may be processed in next step. When all of the tasks that remain are 'delayed' tasks, they are processed normally as in the basic algorithm. This is to ensure that eventually all the tasks are processed.

For the probabilistic approach, each task is mapped into a value in the range of [0:1] according to the function

$$p = \epsilon + (1 - \epsilon)e^{-kl}$$

where l is the length of the search tour (edge traversals) and k is a tuning parameter. For



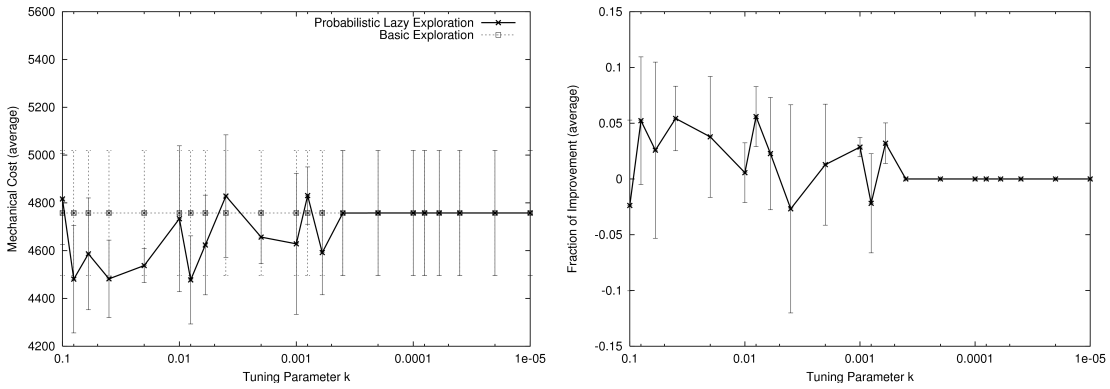
(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.20: Deterministic lazy exploration.

a certain l, the smaller the k is, the bigger the mapped number p is (when k approaches 0, p approaches 1). For a certain k, when l is large, the mapped value p is small, whereas when l is very small (approaches 0), p is high (approaches 1). The robot selects the first task for which a random number r in the range of [0:1] has the property $r \leq p$. Assuming the fairness of random numbers generation, the shorter the search length, the larger p is, and the more likely that the task will be performed. Note that all the tasks are eventually processed.

In the experiments a single robot explores a lattice graph with holes (10×10 lattice with 20% vertices randomly removed), using both the deterministic and the probabilistic approach. For the deterministic approach, tests are conducted with different length thresholds. The test for each sample threshold was repeated 10 times and the average cost and relative improvement are presented in Figure 4.20. Note that in the experiments when the threshold is sufficiently large ($\geq$80), all tasks will be accepted, turning the lazy exploration into the basic brute force strategy (horizontal line in the curves). For the probabilistic approach, tests were conducted with different tuning parameter k, which



(a) Average mechanical cost

(b) Average improvement (fraction)

Figure 4.21: Probabilistic lazy exploration.

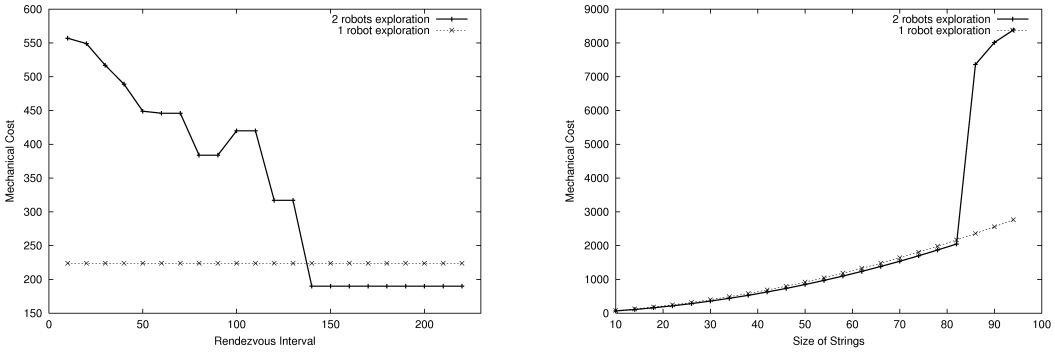
determines the evaluation of the difficulty of tasks. For each sample k the test was repeated 10 times and the average cost were presented in Figure 4.21. Note that when k is sufficiently small ($k \leq 0.0005$), p for all the tasks will approach 1, so all the tasks will be accepted, again turning the lazy exploration into the brute force strategy (horizontal line in the curves).

From the results we can see that the performance of both the approaches depends critically on the factors that determine the acceptance or delaying of the search tasks (threshold and k). Both approaches demonstrate conditions where the performance is worse than the basic algorithm. How to tune the factors for the approaches so that better performance can be achieved is an interesting direction for future research. On average the probabilistic approach is more promising, due to the fact that the majority of its parameter k leads to positive improvements, whereas for the deterministic approach only half of the thresholds enable it to produce positive improvements.

4.6 Strategic rendezvous scheduling

The enhancements described above address the exploration and merging process of multiple robot exploration. Previous experiments show that other than the details of the two processes themselves, the rendezvous schedule in the algorithm is another factor that determines the performance of the multiple robot exploration algorithm. The rendezvous schedule determines when, where and how frequently the merge process should be performed. The rendezvous schedule affects the algorithm's performance and in some cases this influence is critical. Consider the exemplified scenario in which two robots robot_1 and robot_2 explore a string, starting from the middle point and exploring in opposite direction. After a certain interval the robots return for merging. Suppose we want to merge robot_2's map into robot_1's map. Consider two intervals k_1 and k_2 where k_1 is short so when the robots return for merging, robot_1 has not finished exploring its assigned half string (so there are unexplored edges at the very end of its partial representation) and k_2 is a longer interval that allows robot_1 to fully explore its assigned half string (so there is no unexplored portion in its partial representation). Clearly the performance of the multiple robot algorithm at these two intervals will be substantially different. With an unexplored portion at the very end of its partial representation, disambiguation of each place in robot_2's partial representation requires the robot to move to the very end, whereas with a fully explored partial representation, merging of the two maps can be conducted without any disambiguation cost.

Experiments were conducted to investigate the effects of different rendezvous on exploring strings. In the first experiment two robots explore a 20 nodes string with varying rendezvous intervals, in another experiment the rendezvous interval is fixed but robots explore on varying sized strings. Results of both experiments are shown in Figure 4.22. Both the results show that in this exemplified scenario, rendezvous intervals that leave unexplored portions in the base map produce much higher cost than the intervals that al-



(a) Fixed string (20 nodes) with varying rendezvous interval

(b) Varying sized strings with fixed rendezvous intervals (2000 steps)

Figure 4.22: Effects of rendezvous intervals on exploring strings.

low the base map to be fully explored, and the cost is higher than single robot exploration as well.

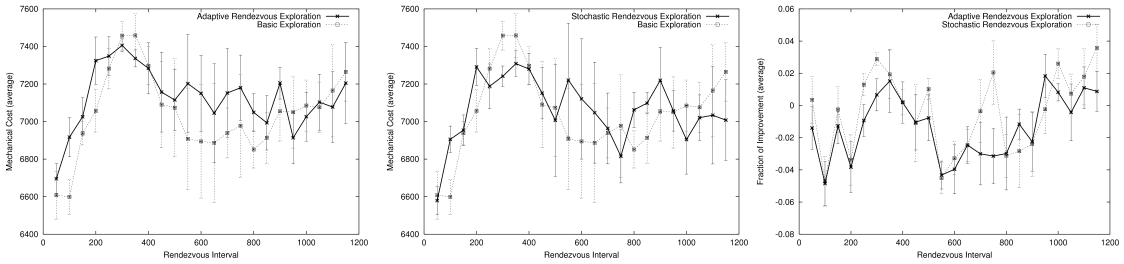
In the basic multiple robot exploration algorithm, robots return to the rendezvous place to merge their partial representations at a fixed interval, e.g., after every 50 mechanical steps. A fixed rendezvous is not topology specific and does not reflect current progress. The enhancement here adopts an adaptive rendezvous schedule in which the rendezvous interval is adjusted after each rendezvous, i.e., to adaptively decide the next rendezvous interval based on the results from previous intervals. Specifically, at a rendezvous using rendezvous interval k_t, robots meet and merge their partial maps, then based on the merge result, robots reason about 'what if we met in some k_t' $(0 < k_t' \leq k_t)$ steps?' Such a question is 'electronically' solvable, due to the fact that the robot can retrace its motions and perceptions at previous steps, and the fact that after doing the merge, robots have the necessary matching information (for all edges and vertices) between the two maps. Based on defined metrics, evaluate the result for some 'virtual'

interval $k_t^{'}$ ($0 < k_t^{'} \leq k_t$). Then based on the evaluation derive the next rendezvous interval for the robots.

Correctness This enhancement does not change the details of the exploration process and the merging process. When incorporated into the basic algorithm, the correctness of the algorithm is not violated.

Evaluation and performance The enhancement does not change the details of the exploration process and merge process. The 'adaptive reasoning' process does not incur extra mechanical cost. So the evaluation for the multiple robot exploration algorithm with adaptive rendezvous scheduling is the same as that for the basic multiple robot exploration algorithm (where fixed rendezvous scheduling is used), as given by Equation 3.2.

Experiments were conducted to evaluated the performance of a simple version of adaptive rendezvous scheduling approach. In this version of adaptive rendezvous approach, the robots are given an initial rendezvous interval k_0, which is used for the first rendezvous. After each rendezvous using k_t, the robots reason about merge results at two 'virtual' rendezvous interval $k_t^{'}$ and $k_t^{''}$, where $k_t^{'} = \frac{1}{3}k_t$ and $k_t^{''} = \frac{2}{3}k$. Then the growth of the resulting (merged) graph evaluated in terms of the edge growth rate at the virtual interval $k_t^{'}$, $k_t^{''}$ and the real interval k_t are compared. If $k_t^{'}$ or $k_t^{''}$ has the highest growth rate, then $k_t^{'}$ or $k_t^{''}$ is used as the next rendezvous interval k_{t+1}. If the growth of resulting graphs shows a consistent growing pattern, i.e., growth rate at the real interval k_t is the highest and $k_t^{'}$ is the lowest, then a longer rendezvous interval $k_{t+1} = \alpha k_t$ ($\alpha > 1$) is used as the next interval. So depending on the growth pattern of the resulting graphs, the next interval k_{t+1} can be either equal to, greater than, or smaller than k_t. To better study the effectiveness of the adaptive approach, we conducted multiple robot exploration using this approach on both nearly homogeneous graph (lattice with 1% vertices removed)



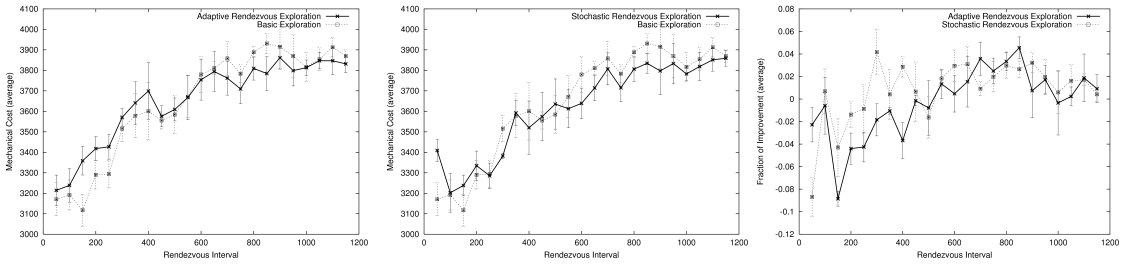
(a) Average cost of adaptive algorithm

(b) Average cost of stochastic algorithm

(c) Improvements of stochastic and adaptive algorithm

Figure 4.23: Adaptive and stochastic rendezvous scheduling (on lattice with 1% holes).

and less homogeneous graph (lattice with 20% vertice removed). Clearly an effective approach is expected to perform better in homogeneous graphs than in hetergeneous graphs. Moreover, we implemented a simple random rendezvous approach and compared the performance of the adaptative approach against the random approach, as well as the basic approach where a fixed interval is used. In the random scheduling approach, at each rendezvous, a next interval within a pre-defined range is randomly generated and used. Experiments were conducted using different intervals. For each sample interval, all three approaches were used by two robots exploring the lattice graphs. For each given interval k, the basic algorithm uses it as the fixed interval for all rendezvous, whereas the random algorithm uses it to randomly generate the next rendezvous interval k_t, where $0 < k_t \leq k$. For the adaptive algorithm, k is used as the initial interval only and subsequent intervals are adaptively generated according to the technique described above using $\alpha = 1.5$. The test for each interval is repeated 10 times each with random holes in the graph. Average cost of both the algorithms and average relative improvement of the two new algorithm over the basic algorithm are presented in Figure 4.23–4.24. From the results we can see that the current version of adaptive scheduling approach does not produce consistent improvements over the basic (fixed scheduling) algorithm, both on the nearly homogeneous



(a) Average cost of adaptive algorithm

(b) Average cost of stochastic algorithm

(c) Improvements of stochastic and adaptive algorithm

Figure 4.24: Adaptive and stochastic rendezvous scheduling (on lattice with 20% holes).

graph and the less homogeneous graph. Moreover, the performance of the approach is close to the random approach. This illustrates that the current evaluation and prediction mechanism in the adaptive approach need to be refined. This would be an interesting direction for future work.

4.7 Merging with large groups of robots ($k > 2$)

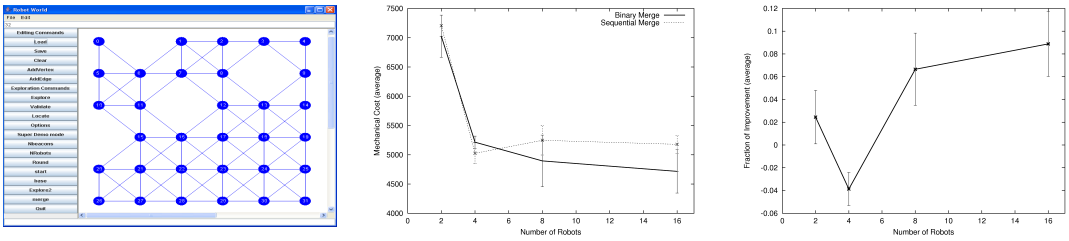
Finally, we look at extending the multiple robot exploration algorithm to the case where there are $k > 2$ robots. When $k > 2$ robots are deployed for exploration tasks, due to the availability of more robots and their markers, there are several stages in the basic multiple robot exploration algorithm where parallelism can be exploited to reduce the task cost and improve the utility of the robots. One such a stage is the merging process. In the basic algorithm the merge order is sequential, i.e., the base robot merges its map with other robots sequentially. The order is arbitrarily determined. To further reduce the task cost and improve the utility of the robots, parallel merge sequences can be explored. For example, one improvement would be to adopt a binary partitioned merge, i.e., while

the 'base' robot[‡] is doing a merge with one of its peers, other pairs of robots merge their representations simultaneously. Initially all of the partial maps are partitioned in binary fashion and pairs of robots are formed to merge their partial maps simultaneously. When all pairs of robots have finished their parallel merge, the resulting maps are partitioned in a binary fashion again and another subphase of parallel merge by pairs of robots starts. The merge phase proceeds with subphases of decreasing number of parallel merges and with the last subphase producing the overall merging result. In each pair of robots doing the parallel merge, one of the robots is designated as the 'local base' robot that performs the merge work as described in the basic merge algorithm.

Correctness This enhancement does not change the details of the exploration process and the merge process. When incorporated into the basic algorithm, the correctness of the algorithm is not violated.

Evaluation and performance Different from the two robots case where the merging process involves one pair of robots doing the merge, in the case of group of robots, a merge phase involves a number of merge processes by pairs of robots. According to the performance metric defined in Chapter 3, for the sequential merge algorithm, the task cost for the merge phase is the sum of the cost of all the merges that are conducted sequentially. Suppose k robots are involved, then a merge phase M_i would have $k-1$ sequential merge processes, each involves the base robot and one of its peers. Denote the j'th $(0 < j \leq k-1)$ merge process in M_i as M_{ij}, then the cost of the merge phase is $cost(M_i) = \sum_{j=1}^{k-1} cost(M_{ij})$, where $cost(M_{ij})$ is given in Equation 3.2. For the binary merge algorithm, the cost is the sum of the cost of all parallel merge subphases by pairs of robots. Assume that merge phase M_i has m parallel subphases, and in the j'th $(0 < j \leq m)$ parallel subphase, denoted M_{ij}, there are $\alpha(M_{ij})$ pairs of robots doing

[‡]The concept of 'base robot' becomes vague and less meaningful in the parallelized merge.



(a) Dense lattice with holes

(b) Average mechanical cost

(c) Improvement over sequential merge

Figure 4.25: Binary vs. sequential merge (dense lattice with holes, up to 16 robots).

parallel merge and denote the l'th $(0 < l \leq \alpha(M_{ij}))$ parallel merge process as M_{ijl}. Then the cost in each subphase M_{ij} is given by

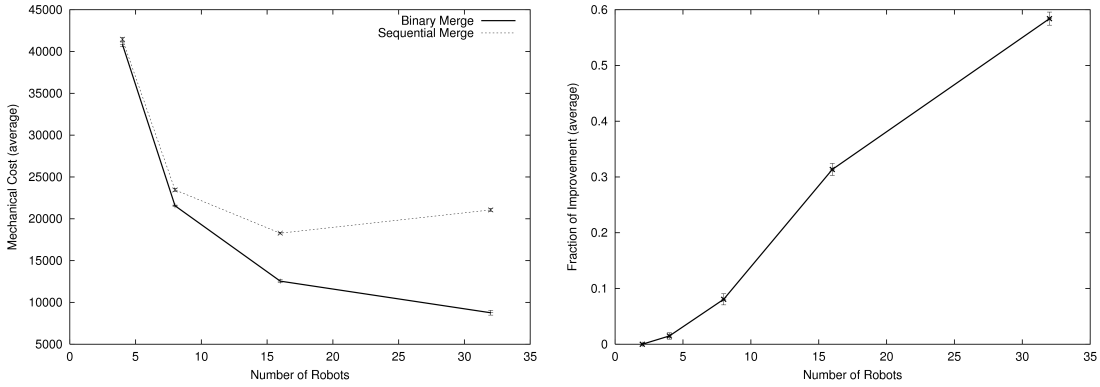
$$cost(M_{ij}) = \max_{l=1}^{\alpha(M_{ij})} \{cost(M_{ijl})\}$$

and the cost of merge phase M_i is given by

$$\begin{aligned} cost(M_i) &= \sum_{j=1}^{m} cost(M_{ij}) \\ &= \sum_{j=1}^{m} \max_{l=1}^{\alpha(M_{ij})} \{cost(M_{ijl})\}. \end{aligned}$$

where $cost(M_{ijl})$ is given in Equation 3.2.

Experiments were conducted to evaluate the performance of one version of the binary merge approach. In this version of binary merge approach, the robot pairs at each merge phase are formed randomly. We compare the algorithm against the basic algorithm, where the merge is performed sequentially and the sequence is arbitrary too. In doing the experiments our concerns include whether using more robots produces better performance,

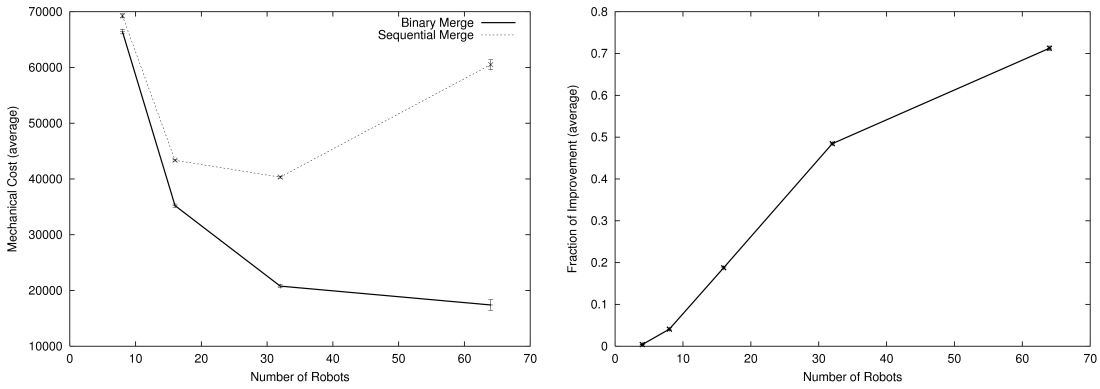


(a) Average mechanical cost

(b) Average improvement over sequential merge

Figure 4.26: Binary vs. sequential merge (55 node regular graph, up to 32 robots).

and whether the binary merge algorithm produces better performance than the sequential merge algorithm. To accommodate groups of robots, we conduct experiments on densely connected graphs. We first run the algorithms on a densely connected lattice with holes in which each vertex is connected to all of its eight 'neighbors' (Figure 4.25(a)). Experiments were conducted using different number of robots, each using both the sequential merge and the binary merge algorithm. Each condition was repeated 10 times and the average costs of both the algorithms and the relative improvement of the binary merge over the sequential merge (on different number of robots) are presented in Figure 4.25(b)–(c), along with standard errors. Results show that for both algorithms, increasing the number of robots from 2 to 4 produces substantial reduction in the cost. To operate with larger groups of robots, we conducted experiments on more densely connected graphs – regular graphs. Experiments were conducted using both algorithms on a 55 node regular graph with up to 32 robots, and then on 80 node regular graph with up to 64 robots. Each condition was repeated 10 times and the average cost and relative improvement are shown in Figure 4.26-4.27. We can see that with regular graphs, as the number of robots



(a) Average mechanical cost

(b) Average improvement over sequential merge

Figure 4.27: Binary vs. sequential merge (80 node regular graph, up to 64 robots).

increases, the binary merge algorithm provides a consistent reduction of cost, and this is not true for the sequential merge algorithm. Moreover, the binary merge algorithm starts to produce a substantial improvement when 16 robots are deployed, and as the number of robots increases, the improvement increases. We also conducted experiments on less homogeneous graphs. The binary merge algorithm provides an improvement but does not show a consistent pattern as in the regular graph case.

4.8 Summary

This chapter presented a number of potential enhancements that can be made to both the original solution of Dudek et al. [14, 15] as well as to portions of the basic multiple robot algorithm described in Chapter 3. Some strategies address the exploration process, such as breadth-first exploration and lazy exploration; some strategies address the merging process, such as using multiple robots. Some strategies can be used to improve both the exploration and the merging process, such as exploiting extended vertex signatures and

Portions addressed / Enhancements	Single robot exploration	Multiple robot exploration	
		Exploration phase	Merge phase
Exploiting neighbor information	✓	✓	✓
Exploiting communication information		✓	✓
Using multiple robots during merging			✓
Breadth-first exploration	✓	✓	
Lazy exploration	✓	✓	
Strategic rendezvous scheduling		✓	
Merging with groups of robots		✓	

✓ portion of the algorithm that the enhancement addresses

Table 4.1: Enhancements and portions of the problem they address.

exploiting communication when robots encounter each other during exploration. Other than the exploration and merging process themselves, we also investigated enhancements that address other portions of the problems, such as strategic rendezvous scheduling and dealing with large group of robots. Table 4.1 presents a summary of the portions of the problem into which each of these enhancements can be incorporated. For each enhancement, the corresponding portions of the problem they address are indicated.

5 Summary and future work

This report investigates the problem of exploration and mapping in an embedded graph-like world where a collection of identical robots are deployed. This report formally extends Dudek et al.'s single robot exploration algorithm to the case of multiple robots. By equipping the robot with abilities to disambiguate places, Dudek et al.'s approach is a deterministic solution to the topological SLAM problem. Building upon the multiple robot exploration algorithm presented in [20], a multiple robot exploration algorithm has been developed which enables a group of two or more robots to construct a topological map isomorphic to the underlying world being explored. An evaluation metric for multiple robot exploration is developed as well.

This report also explores a number of potential enhancements that can be made to both the original solution of Dudek et al. as well as to portions of the multiple robot exploration algorithm. These enhancements address core techniques required in multiple robot exploration, such as location disambiguation, merging partial representations and rendezvous scheduling.

Empirical evaluation of both the basic and enhanced multiple robot exploration algorithms has shown that two robots can explore an environment much more efficiently than a single robot, and that super-linear performance improvements can be encountered. This is due to the fact that many environments have natural decompositions that enable multiple robots to operate independently and many of the mechanically complex tasks can be reduced through the use of multiple robots.

The work presented in this report suggests several extensions and variations to the basic exploration algorithm.

5.1 Within the current model

Within the current model, issues deserving further investigation are related to both the enhancements described in Chapter 4 and other portions of the algorithm.

Extending current enhancements

Extensions and variations to the enhancements described in Chapter 4 suggest several possible directions for future research.

Extended signatures In this report we evaluated the extended signatures with neighborhood radius r of up to 2. Further studies should be given to a broader view of signature, i.e., signatures that includes the topology information of a larger neighborhood. A broader view of node signature is expected to produce a better disambiguation ability than the signatures considered here. Used in the merge phase, this better disambiguation ability comes at an increased computational cost, while in the exploration phase it comes at both increased computational cost (for signature comparing) and additional mechanical cost (for signature retrieval). It would be interesting to investigate issues related to a broader view of signatures, e.g., what is the tradeoff of the increased disambiguation ability and the increased computational and mechanical cost? Does the disambiguating ability stop increasing after a certain neighborhood distance? An alternative to the broader view of signatures is to develop a narrower view of signature that considers partial neighbor topology information. Signatures based on partial neighbor topology might produce impaired disambiguation ability but require reduced computational and mechanical cost. A narrower view of node signature might be appealing in the

exploration phase, where the extra cost required for retrieving neighbor information of each unknown place depends on the number of neighbors (edges) of the unknown place. It's worthwhile investigating the tradeoff of the decreased disambiguation results and the decreased computational and mechanical cost.

Communication during exploration In addition to the two opportunistic encounter scenarios described in this report, there exist other encounter situations in which communication information could be exploited. For example, when two robots encounter each other in a common vertex and both of them are about to explore the same unexplored edge of the common vertex, one of the robots (e.g., the one that requires a longer search tour) can terminate this task to avoid duplicate work. Another useful communication would be to provide updates to the partial maps. When two robots encounter each other, they could examine their current partial maps to see if some new edges connecting the commonly known vertices (i.e., direct edges) have been added in one map but not in the other map. If this is the case, instead of waiting until the merge phase, these edges can be added to the corresponding maps immediately (by direct fusing). With more edges in the maps, both the ongoing exploration phase and subsequent merge phase are expected to be facilitated. Since direct fusing does not require mechanical costs, the overall task cost is expected to be reduced.

Exploiting multiple robots in the merge phase We described an enhancement that exploits the existence of multiple robots in the back-link validation stage of the merge phase. There are other stages in the merge process in which another robot could be helpful. One such a stage is the search stage. As an example, one approach to utilizing multiple robots in search stage would be to let the other robot help the base robot by visiting some of the potentially confusing vertices in parallel. Compared with the enhancement in the back-link validation stage, parallel search requires more sophisticated

coordination between the robots. There are some challenging issues that need to be addressed, e.g., how to partition the search route so that the cost of the parallel search is reduced, both in the case that the marker is found and the case that it is not found. One approach to partitioning the search route would be to partition it as equally as possible. This approach, however, may not be optimal if the marker is found (so the partitioned search route is not fully followed). If the search routes are largely disjoint, it would be a challenge for the robot that found the marker to inform the other robot. An approach that tries to address this problem would be to partition the routes so that the routes intersect as much as possible. This approach facilitates communication between the robots but it may not be optimal if the marker is not found. An alternative approach would be to let the robots come back to the starting location regularly so in case the marker is found, the robots meet early (and therefore the search can terminate early). Obviously the approach might incur significant extra traversals to the starting location. Developing an efficient parallel search strategy is a challenging but interesting direction for future work. An evaluation mechanism for the parallel search algorithm needs to be carefully defined as well.

Breadth-first exploration We studied the performance of the breadth-first exploration for a lattice graph with holes but performance of the algorithm on more heterogeneous graphs should be studied further. We studied the performance of breadth-first exploration for single robot exploration. Integrating breadth-first exploration in the exploration phase of multiple robot exploration is worth investigating.

Lazy exploration In the current lazy exploration a delayed task is 'released' once some other task is accepted and processed, i.e., delayed tasks are considered again in the next round of selection. It would be interesting in the future to investigate the effects of delaying a hard task for later so that when it is considered again, the explored graph

might have grown more.

Integrating lazy exploration into the exploration phase of multiple robot exploration would be another interesting direction for future work. In multiple robot exploration, there are more 'opportunities' to further delay a difficult task. For example, the unknown place may be solved with fewer steps by the other robot; the unknown place may be solved in the following merge phase, or even in later exploration and merge phases with fewer cost, due to the growth of the explored graph.

Adaptive rendezvous scheduling As mentioned earlier, due to the unsatisfactory performance of the current adaptive rendezvous scheduling algorithm, more sophisticated evaluation and predication techniques should be explored to reflect the evolvement pattern of the resulting graph more accurately. The current adaptive scheduling algorithm uses virtual intervals to evaluate and predict the growth pattern of the merged graph. Techniques based on more (or even all) possible virtual rendezvous intervals should be developed to better reflect the growth pattern of the resulting graph. Currently the growth of the resulting graph is evaluated in terms of its edge growth rate (over mechanical steps). This edge growth rate, however, might not fully represent the growth of the resulting graph. Earlier experiments show that the growth in the number of vertices of the resulting graph is not always consistent with the growth of the number of edges. A better evaluation metric should be developed.

Other adaptive rendezvous scheduling strategies should be developed. For example, one straightforward strategy would be that robots perform their rendezvous when they encounter each other on their way back to the rendezvous place. Extending this idea, another strategy would be that robots perform rendezvous whenever they encounter each other during independent exploration.

A rendezvous schedule includes both the rendezvous interval and location. The cur-

rent version of adaptive rendezvous scheduling algorithm does not address the issue of rendezvous location. As in the use of a fixed rendezvous interval, a fixed location does not reflect the progress of the exploration so it is worthwhile exploring approaches in which rendezvous locations are adaptive as well. While reasoning about the rendezvous intervals, the robots might also reason about rendezvous locations based on additional virtual locations selected from the set of the commonly known locations.

Strategic merge for group of robots In the binary merge algorithm, the pairs of robots are formed arbitrarily. How to form pairs of robots more strategically is a challenging problem worth investigating. One heuristic is to form pairs of robots in such a way that maps within the pair are 'close' so traversals during the merge might be low. A well defined metric is required to measure the 'distance' of two partial maps.

Other than the merge sequence, with group of robots the merge process itself can also be improved. Similar to the earlier enhancement in which communication information is exploited when robots encounter, one strategy would be to 'dispatch' some robots to some of the vertices in the base map which have unexplored edges and leave the robots (or their markers) there. As a vertex in another map (the exterior end of the current frontier) is selected and visited by the base robot, if the vertex is in fact the counterpart vertex of some vertex in the base map, the base robot might 'encounter' one of the dispatched robots, or one of their markers. As in the earlier enhancement, subsequent search in the base map for disambiguation is then unnecessary and back-link validation can start. This strategy is appealing when the number of robots involved becomes a large fraction of the number of vertices in the base map that need to be disambiguated and merged.

Other enhancements

In addition to extending the current approaches, there are some other portions of the problem that suggest directions for future research.

Task division How to partition the unexplored portion of the merged graph is a critical technique that is worthy of further investigation. In this report a simple strategy is used. This strategy partitions the unexplored edges of the merged graph between the robots evenly. It would be interesting to fully explore the task division issue in multiple robot exploration and develop strategic techniques to partition the exploration task. We described above a possible enhancement in which encountering robots might avoid exploring the same edge. A dynamic partition approach would be to have the robots re-partition the currently unknown portions of the world when they encounter each other during independent exploration.

Integration of current enhancements Most of the enhancements so far have been evaluated independently. It would be interesting to integrate these enhancements and investigate the performance of the resulting algorithm. Some enhancements can be integrated trivially. For example, a multiple robot exploration algorithm could use adaptive rendezvous scheduling; robots could communicate when they encounter each other during exploration; in the merge phase, partial matching information (from communication during exploration) and extended signatures could be used to save mechanical traversals, and when the base robot moves to disambiguate a vertex, the other robot could remain at the other end of the edge to generate a unique signature of the other end of the link.

It would be interesting to integrate other enhancements in a strategic way. For example, an algorithm may integrate both depth-first exploration, breadth-first exploration, extended signature exploration and lazy exploration strategies. Then during the explo-

ration these strategies are selected in a case by case manner. For example, the robot might reason about the homogeneity of the current graph and when the graph is evaluated as being relatively homogeneous, use breadth-first search. When the current area is relatively heterogeneous, extra mechanical moves could be used to retrieve an extended signature.

More performance concerns Throughout this report, the task cost is the main concern. It would be interesting to investigate other costs involved in completing an exploration task, such as the utility of the robots, total mechanical cost, and even electronic (computational) cost. Utility of the robots measures how well the robots are utilized in the process of completing the task. In the current model, utility can be measured in terms of the fraction of a robot's mechanical cost over the overall task cost of completing the task. Developing algorithms that consider both the task cost and utility leads to several extension and variations. For example, the adaptive rendezvous scheduling might also consider the utilization of the robots at each virtual interval and location, and the next rendezvous schedule chosen would be the one that produces both low task cost and high robot utilization. Another example would be that during independent exploration, when a working robot encounters another robot who has finished its exploration and is waiting, the waiting robot might take up some of the unfinished work of the working robot. More aggressively, when a robot finishes its exploration earlier, instead of waiting at the rendezvous place, in order to help the other robot, it might traverse the graph trying to meet the other robot.

5.2 Beyond the current model

Extensions and variations of the current model also suggest a number of directions for future work.

More perception power It would be interesting to investigate how enhanced perception power improves the performance of robotic exploration. For example, a model under which the robot can see which edge another robot entered the vertex from would provide enriched information that can be exploited. With this enhanced perception model, when robots encounter each other during their exploration, instead of the current partial matching information (vertex label), full matching information (both vertex label and local edge index correspondence) might be generated, which might further facilitate the ongoing exploration and subsequent exploration and merge phases.

More communication power Enhanced communication power also suggests several problems worth addressing. For example, assume robots can sense each other when they are within a distance of d edges of the vertex. How does the communication range d affect performance?

Global clock A global clock is another powerful source of information that can be utilized to improve exploration. With a global clock, more communication information can be exploited. Suppose we let each robot record the global time and location of each of its marker operation (dropping and picking up). Each robot also records the global time and location if it sees a marker of the other robot. When the robots encounter each other, by examining the marker related records of the robots, more partial matching information of vertices can be generated. Suppose the records show that one robot dropped its marker at global time 10 at local vertex v_1 and picked it up at time 20, and another robot saw the marker at global time 15 at local vertex v_2. Then it can be inferred that v_1 and v_2 are counterpart vertices.

Marker related extensions Finally, without going into details, we see that there are marker-related extensions to the current model that suggest directions for future work,

e.g., assuming each robot has multiple markers, assuming markers that can point in a direction, or markers that can be dropped on edges etc. For example, with an directional marker that points to the direction (exit) by which the robot came to drop the marker, the back-link validation (when the marker is found) would be facilitated.

Bibliography

[1] T. Arai, E. Pagello, and L. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on Robotics and Automation*, 18(5):655–661, 2002.

[2] T. Balch and R. C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

[3] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa. Two applications of complementation via inductive counting. Technical Report no. 58972, IBM Research Division, 1987.

[4] R. Brooks. A robust layered control system for a mobile robot. Technical Report AIM-864, MIT AI Lab, 1985.

[5] H. Bunke. Graph matching: Theoretical foundations, algorithms, and applications. In *International Conference on Vision Interface*, pages 82–88, Montreal, Quebec, Canada, 2000.

[6] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 476–481, San Francisco, CA, USA, 2000.

[7] A. Burns and G. Davies. *Concurrent Programming.* Addison-Wesley Publishing, Co., USA, 1993.

[8] Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

[9] M. Chandy and S. Taylor. *An Introduction to Parallel Programming.* Jones and Bartlett Publishers, Inc., USA, 1992.

[10] E. Davis. *Representing and Acquiring Geographic Knowledge.* Morgan Kaufmann Publishers Inc., USA, 1986.

[11] G. Dudek, P. Freedman, and S. Hadjres. Using local information in a non-local way for mapping graph-like worlds. In *13th International Conference on Artificial Intelligence*, pages 1639–1647, Chambery, France, 1993.

[12] G. Dudek, P. Freedman, and S. Hadjres. Mapping in unknown graph-like worlds. *Robotic Systems*, 13(8):539–559, 1998.

[13] G. Dudek, M. Jenkin, and E. Milios. A taxonomy of multirobot systems. In T. Balch and L. Parker, editors, *Robot Teams: From Diversity to Polymorphism*, pages 3–22. A.K. Peters, Natick, Massachusetts, 2000.

[14] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. Technical Report RBCV-TR-88-23, Research in Biological and Computational Vision, Department of Computer Science, University of Toronto, 1988.

[15] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 6(7):859–865, 1991.

[16] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Map validation and self-location in a graph-like world. In *13th International Conference on Artificial Intelligence*, pages 1648–1653, Chambery, France, 1993.

[17] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robust positioning with a multi-agent robotic system. In *IJCAI-93 Workshop on Dynamically Interacting Robots*, pages 118–123, Chambery, France, 1993.

[18] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for swarm robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 441–447, Yokohama, Japan, 1993.

[19] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, 1996.

[20] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Topological exploration with multiple robots. In *7th International Symposium on Robotics with Application (ISORA)*, Anchorage, Alaska, USA, 1998.

[21] H. Durrant-Whyte. Uncertainty geometry in robotics. *IEEE Journal of Robotics and Automation*, 4(1):23–31, 1988.

[22] A. Elfes. Sonar based real world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987.

[23] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, 1989.

[24] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(5):2015–2028, 2004.

[25] J. Fenwick, P. Newman, and J. Leonard. Cooperative concurrent mapping and localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1810–1817, Washington, DC, USA, 2002.

[26] J. Fitzpatrick, D. Hill, and C. Maurer. Image registration. In M. Sonka and J. Fitzpatrick, editors, *Handbook of Medical Imaging, Volume 2: Medical Image Processing and Analysis*, pages 1–35. SPIE Press, Bellingham, WA, USA, 2000.

[27] D. Fox, S. Thrun, F. Dellaert, and W. Burgard. Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 499–516. Springer Verlag, New York, 2001.

[28] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotic Research*, 23(9):939–954, 2004.

[29] T. Gibb. Quecreek commission says better maps are a must. Pittsburgh Post Gazette, November, 2002.

[30] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[31] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2121–2427, Las Vegas, Nevada, USA, 2003.

[32] A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[33] W. Huang and K. Beevers. Topological map merging. *International Journal of Robotics Research*, 24(8):601–613, 2005.

[34] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[35] B. Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[36] B. Kuipers and Y. Byun. A qualitative approach to robot exploration and map-learning. In *Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pages 390–404, St. Charles, IL, USA, 1987.

[37] B. Kuipers and T. Levitt. Navigation and mapping in large-scale space. *AI Magazine*, 9(2):25–43, 1988.

[38] L. Matthies and S. Shafer. Error modelling in stereo navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, 1987.

[39] M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping*. PhD thesis, Carnegie Mellon University, 2003.

[40] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[41] S. Norton and M. DiPasquale. *Thread Time: The Multithreaded Programming Guide*. Prentice Hall PTR, USA, 1997.

[42] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

[43] E. Pagello, A. D'Angelo, F. Montesello, F. Garelli, and C. Ferrari. Cooperative behaviors in multi-robot systems through implicit communication. *Robotics and Autonomous Systems*, 29(1):65–77, 1999.

[44] L. Parker. Current state of the art in distributed autnomous mobile robotics. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 3–14, Knoxville, TN, USA, 2000.

[45] F. Preparata and M. Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

[46] I. Rekleitis, G. Dudeck, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *International Joint Conference on Artificial Intelligence*, pages 1340–1345, Nagoya, Japan, 1997.

[47] I. Rekleitis, G. Dudek, and E. Milios. Accurate mapping of an unknown world and online landmark positioning. In *Vision Interface(VI)*, pages 455–461, Vancouver, Canada, 1998.

[48] I. Rekleitis, V. Dujmovi, and G. Dudek. Efficient topological exploration. In *IEEE Internation Conference on Robotics and Automation (ICRA)*, pages 676–681, Detroit, MI, USA, 1999.

[49] I. Rekleitis, R. Sim, G. Dudek, and E. Milios. Collaborative exploration for the construction of visual maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1269–1274, Maui, Hawaii, USA, 2001.

[50] N. Roy and G. Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, 2001.

[51] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *National Conference on Artificial Intelligence (AAAI/IAAI)*, pages 852–858, Austin, TX, USA, 2000.

[52] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pages 390–404, St. Charles, IL, USA, 1987.

[53] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[54] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[55] S. Thrun. Robotic mapping: a survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millennium*, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[56] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, USA, 2005.

[57] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4270–4275, Taipei, Taiwan, 2003.

[58] S. Thrun and Y. Liu. Multi-robot SLAM with sparse extended information filers. In *The 11th International Symposium of Robotics Research (ISRR'03)*, Sienna, Italy, 2003.

[59] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous mapping and localization with sparse extended information filters. *International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[60] B. Yamauchi. Frontier-based exploration using multiple robots. In *2nd International Conference on Autonomous Agents*, pages 47–53, Minnepolis, MN, USA, 1998.