EXPLORING TOPOLOGICAL ENVIRONMENTS

HUI WANG

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
OCTOBER 2014

Abstract

Simultaneous localization and mapping (SLAM) addresses the task of incrementally building a map of the environment with a robot while simultaneously localizing the robot relative to that map. SLAM is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robots. This thesis considers the SLAM problem within a topological framework, in which the world and its representation are modelled as a graph. A topological framework provides a useful model within which to explore fundamental limits to exploration and mapping. Given a topological world, it is not, in general, possible to map the world deterministically without resorting to some type of marking aids. Early work demonstrated that a single movable marker was sufficient but is this necessary? This thesis shows that deterministic mapping is possible if both explicit place and back-link information exist in one vertex. Such 'directional lighthouse' information can be established in a number of ways including through the addition of a simple directional immovable marker to the environment. This thesis also explores non-deterministic approaches that map the world with less marking information. The algorithms are evaluated through performance analysis and experimental validation. Furthermore, the basic sensing and locomotion assumptions that underlie these algorithms are evaluated using a differential drive robot and an autonomous visual sensor.

Acknowledgements

I extend my sincere appreciation and gratitude to many people who made this work possible.

Firstly, I am especially grateful to my supervisors Professor Michael Jenkin and Professor Patrick Dymond, whose intellectual rigor and insight have been a constant source of encouragement and inspiration. They are always available, always insightful, and, always patient. They provided me with invaluable advice and comments on both my research and my future career plans. I felt honored to study under their supervision.

I would also like to extend my sincere gratitude to the rest of my supervisory and examining committees for their support. Thank Professor Eric Ruppert for providing me with invaluable advice and comments on the thesis. Thank you for guiding me on formalizing the arguments in the thesis. Thank Professor Evangelos E. Milios for providing valuable and detailed comments on every page of the thesis. Thank you for your valuable time. Thank Professor John E. Moores for inspiring me on exploring the practical applications of the theoretical results of the thesis. Thank Professor Zbigniew Stachniak for your support, encouragement and inspiration.

Finally, my eternal gratitude gives to my family for their never-ending support and love. I would like to dedicate this work to all of you.

Table of Contents

Chapter 1

Introduction

Robotic exploration and mapping addresses the problem of acquiring a spatial model (a 'map') of a physical environment autonomously. This problem is one of the most important problems in the pursuit of building truly autonomous robot systems, and is thus a problem of significant practical importance. If robots are to operate autonomously in 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. Even in safer and simpler environments such as the interiors of buildings, accurate mapping of the environment is important. 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. Without a map, many robotic tasks become difficult or even impossible.

Acquiring maps with mobile robots is a challenging problem for a number of reasons. A critical challenge arises from the fact that a robot must represent (localize) itself within the map as the map is being constructed. Constructing a map requires a 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 termed *SLAM*, which

is short for *Simultaneous Localization and Mapping*. SLAM is considered to be a challenging yet fundamental problem in robotics. Robust solutions to SLAM enable a wide range of other robotic tasks which can then assume a common representation within which planning, sensing and action can be performed.

One partitioning of approaches to the SLAM problem is into those that rely on and construct a topological representation and those that use a metric one. While metric representations capture the metric properties (e.g., Cartesian coordinates) of the environment, topological (graph-like) representations describe the connectivity of different places. A topological (graph-like) representation represents the basic information that a robot must be able to represent in order to distinguish one place from another. By abstracting away many of the details necessary for SLAM within a metric formalism, a topological representation provides a useful theoretical model within which to explore fundamental limits to exploration and mapping.

Answering the question 'have I been here before?' is a core problem that lies at the heart of SLAM algorithms. This problem is also known as 'loop closing' because identifying that 'I have been here before' enables loops to be constructed or closed in the partially built map. There are two fundamental approaches to loop closing. One approach is to exploit metric information. In this type of approach, the robot's pose and motion measurements are captured, resulting in a representation of the world that captures metric properties of the environment. Within the metric formalism, a common approach is to cast the problem within a probabilistic framework, which models the different sources of uncertainties of the metric information. Effective strategies have been developed that utilize Bayesian methods to integrate measurements and estimates with

appropriate probability distribution functions. Specifically, the task is modelled 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. The Kalman filter [56, 39] and the particle filter [56, 35] are important techniques that have been used successfully to approximate the SLAM posterior. In general, the probabilistic approaches to SLAM with metric information work well under some reasonable assumptions about environmental complexity. As the environment becomes bigger, however, the approaches are challenged with issues such as computational complexity and data association. Also note that metric solutions ignore the power of more discrete and sparse representations.

Another approach to SLAM maintains a topological representation and many such solutions resort to an external 'marking aid' to help the robot solve the loop closing problem deterministically. Various kinds of markers have been explored in recent robotics literature, in which 'markers' are also known as 'pebbles', 'tokens' or 'beacons'. Equipped with an appropriate marker or collection of markers, an explorer can solve the SLAM problem deterministically. One kind of marker that has been examined extensively is a unique movable marker that can be dropped and picked up by the robot during exploration. Previous work including [26, 27, 19, 17] examined the power of movable markers in exploring undirected graph-like worlds. It is shown in [26] that a single movable marker that can be dropped and picked up at vertices is sufficient to solve the SLAM problem deterministically. Another kind of marking aid that has been examined extensively utilizes sufficient distinct markers which the robot can use to uniquely mark each place it visits in the graph-like world. (In these papers, the mapping problem is modelled as exploring a *labelled*

graph [46, 21, 45].) There are a number of problems related to marker-based exploration within the topological formalism that have not been fully addressed in the literature. One fundamental problem is: what is the minimal marker information that is required for deterministic SLAM? Can a marker that is 'weaker' than a movable marker, such as a single immovable marker, be used to solve the SLAM problem deterministically, as is the case for a movable marker? Other interesting questions include: are there efficiencies to be found by using different forms of marker classes such as edge markers, directional (vertex or edge) markers, thread-based markers and the like, what advantages are there in the use of multiple immovable markers, and what kind of marker can result in optimal cost?

The success of probabilistic representations in metric mapping suggests the potential for probabilistic approaches within the topological representation as well. Some preliminary research on marker-less exploration within a topological formalism exists. For example, [22, 23] propose a probabilistic marker-less approach which does not resort to the use of markers or metric information. Without a marker aid or metric information, this approach is based on the structure of the world itself. In contrast to the marker-based approaches where a single unique solution (map) is produced, here multiple models (hypotheses) of the unknown world may result. The multiple world models are consistent with the robot's (impoverished) observations, but only one is the correct interpretation of the real world model. Unfortunately, infinitely many world models may result, as there may be no way of collapsing the probability function without resorting to other external aids. The challenges faced by the approach indicate the importance of fully utilizing the structure of the environment, and the importance of utilizing other additional information about

the environment, which can be used as a cue to cease exploration even though some possible models have not been fully explored.

Objectives and contribution of the thesis

This thesis considers the SLAM problem within a topological framework. Within the topological formalism, this thesis examines the relative expressive power of different marking aids in solving the SLAM problem deterministically, and it also explores mapping with insufficient marking aids, including approaches that map the world without resorting to marking aids at all.

This thesis contributes to existing research in the following ways.

1. It explores the fundamental information that is required for deterministic topological SLAM. It shows that having both explicit place and back-link information in a single vertex is sufficient to solve the SLAM deterministically. It also shows that such information can be provided in a number of ways including through the addition of a single directional immovable marker in the environment, which can be considered minimum marker in terms of the number of markers and the amount of robot operations on the marker.

    - This work develops a basic algorithm for mapping using a single directional immovable marker. Rigorous proof and cost bounds are derived.
    - This work validates the basic algorithm using a real robot system, showing that the basic sensing and locomotion models assumed in topological SLAM algorithms are realistic when applied to real-world environments, sensors and robotic platforms.

- This work develops several enhancements that potentially improve on the performance of the basic algorithm.

2. It explores how to map with "less" marker information, and investigates approaches to topological mapping without resorting to a marking aid.

3. It explores the expressive power of several other marker classes that have not been examined in the literature, including single edge marker, multiple immovable vertex and edge markers, and thread-based markers.

Structure of the thesis

This thesis is organized as follows.

- Chapter 2 defines formally the problem addressed in this thesis and surveys related work in the field of robotic mapping and exploration, with a focus on relevant topological approaches. Open problems are also identified.

- Chapter 3 investigates the fundamental information that is required for deterministic topological SLAM, identifying that having both explicit place and back-link information at a vertex is sufficient to solve topological SLAM deterministically. Algorithms for mapping with such information are developed, justified, and evaluated both via simulation and on real robot systems.

- Chapter 4 explores several enhancements to the basic mapping algorithm described in Chapter 3.

- Chapter 5 explores mapping with less marking information, including mapping without resorting to marking aids.

- Chapter 6 examines the expressive power of other types of markers, including single edge marker, multiple immovable vertex and edge markers, and thread-based markers.

- Chapter 7 presents a discussion of possible directions for future work.

Chapter 2

Related work

Solving the robotic exploration and mapping problem in an unknown environment 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?'). When mapping and localization were introduced by researchers in the early 1980's, research focused on solving the mapping and localization problems independently. More recent research efforts address these two problems simultaneously, and is known as SLAM (Simultaneous Localization and Mapping).

The SLAM problem arises when the robot does not have access to a map of the environment, nor does it know its own pose. In such situations the problem of constructing a map of an unknown environment requires the solution to localization (pose estimation), whereas solving localization requires the solution to mapping. In the absence of both an initial map and exact pose information, the problem is challenging.

2.1 Spatial representations

A critical issue for SLAM is how to model the underlying environment. In the SLAM literature the spatial representations are broadly categorized into *topological* representations and *metric* representations. Topological (graph-like) representations describe the connectivity of different places. Metric representations, on the other hand, capture the metric properties (e.g., distances and angles) of the environment. In metric representations every component of the environment is embedded within some Cartesian space. While topological representations are concise, metric representations provide a more detailed world representation. These two paradigms can be considered as the two ends of a space of pose representations. At one end is a (pure) topological representation, and at the other end is an (embedded) metric representation. While topological and metric representations are the two traditional paradigms in the SLAM literature, there exist other representation possibilities. These possible representations include *geometric* representations that lie in between the two extremes and hierarchical representations that integrate aspects of both topological and metric representations. In contrast to the (pure) topological representations, in a geometric representation some geometric information is maintained, but unlike in the (embedded) metric presentations the components are not necessarily embedded within a Cartesian space. In [3] the map consists of collection of segments, and the angles between pairs of segments are maintained. In [38] the map consists of vertices and edges which are annotated with certain geometric information such as path length and relative orientations of incident edges at each vertex.

(a) Feature (landmark) based metric map

(b) Occupancy Grid map

(c) Topological (graph-like) map

Figure 2.1: Sample metric and topological spatial representations.

Metric representations

Metric representations can be further divided into *feature-based* maps and *location-based* maps (see Figure 2.1 for examples). In feature-based maps, the world is represented as a set of spatially located features (landmarks), each with an associated position 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 [32], 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, while the main advantage of a location based map (e.g., occupancy grid map) representation is that it can be used to represent unstructured environments [56].

Topological representations

Another spatial representation that finds wide application in deterministic SLAM algorithms is a topological (graph-like) representation that describes the connectivity of different places.

Depending on the nature of the algorithm, different definitions and implementations of the topological representation exist. Typically, environments in topological maps are represented as an embedded graph where vertices in the graph correspond to significant places in the environment and edges in the graph denote the adjacency relations between the places. There are various ways that the embedding can be defined. For example, in [26] the graph is embedded within some space in such a way that relative directions are defined on the edges incident upon a vertex. In particular, 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 is the graph embedding adopted in this work.) There exist other graph-like representations of the environment as well. For example, in [41] the topological map is represented as a bipartite graph, with vertices corresponding to (both) places and paths, and arcs (edges) corresponding to the assertion that a particular place is on a particular path. Within a topological (graph-like) representation, SLAM can be approached as a graph-theoretic problem, making it feasible to investigate general issues related to robot exploration within this representation. Moreover, the graph-like representation is an abstract view of the environment and consequently requires low space complexity [55]. Topological representations thus can be considered as the basis of finer geometric and metric representations. Results obtained within topological formalisms can often be readily transferred to geometric and metric representations.

Based on the map representation adopted by the algorithm, the field of mapping can be generally divided into *metric* and *topological* approaches. The relevant approaches are reviewed below.

2.2 Metric SLAM approaches

Metric approaches to mapping and localization adopt a metric representation of the space. While it is possible to build a map representation – either metric or topological – that is either deterministic or probabilistic, metric approaches typically use probabilistic concepts to explicitly represent and manipulate spatial uncertainty. In this type of approach, the robot's pose and measurements are associated with metric information, resulting in a representation of the world which captures the metric properties of the environment. A critical issue in metric-based representation is dealing with errors associated with estimates of the robot's pose and measurements. Effective strategies for metric-based SLAM have been developed that utilize Bayesian methods to integrate measurements and estimates with appropriate probability distribution functions. Within a probabilistic framework, the robotic mapping and localization task is usually modelled as inferring a quantity x (map or robot state) from some data d (measurement, control), represented as $p(x|d)$ and referred as the *posterior* or *belief*. Specifically, the solution to the SLAM problem at time step t is modelled as recovering the best estimate of the robot's current pose s_t, and the map Θ, given the set of all (noisy) observations $z^t = \{z_1, z_2, ..., z_t\}$, controls $u^t = \{u_1, u_2, ..., u_t\}$, and possible data associations $n^t = \{n_1, n_2, ..., n_t\}$, which describe the mapping of observations z^t to features in map Θ. Within a probabilistic framework, this is expressed by the probability called the SLAM posterior $p(s_t, \Theta|z^t, u^t, n^t)$. Under the Markov assumption, the posterior can be expressed using

a Bayes' filter as

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta \underbrace{p(z_t | s_t, \Theta, n^t)}_{\text{perceptual model}} \int \underbrace{p(s_t | s_{t-1}, u_t)}_{\text{motion model}} p(s_{t-1}, \Theta | z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1}$$

where η is a normalization factor ensuring that the resulting value is a probability[1].

In general, the posterior cannot be evaluated in closed form. Various solutions to SLAM can be characterized by the way in which they estimate the posterior. The Kalman filter [56, 39] and the Particle filter [56, 35] are important approaches to approximating the SLAM posterior. Kalman filters and extended Kalman filters are Bayes filters that represents the posterior using multivariate Gaussians. The key assumption here is that both the perceptual model and motion model are linear processes with added Gaussian noise. This assumption can be overly restrictive. By representing the belief $bel(x)$ as a set of m weighted samples distributed according to $bel(x)$, the Particle filter is a popular alternative to approximating the posterior. The beauty of this approach is that given enough samples, any probability distribution can be represented. The main disadvantage lies in its computational complexity, due to the potentially huge number of particles that may be needed to represent the distribution. There also exist hybrid approaches that employ elements of both particle filtering and Kalman filtering. Note that these techniques can also be adopted in topological approaches to SLAM. Recent reviews of metric SLAM approaches can be found in [55, 56, 31, 5]. In summary, probabilistic approaches to SLAM with metric information work well under some reasonable assumptions about environmental complexity. As the environment becomes bigger, however, the approaches are challenged with issues such as

[1] A complete derivation of this expression can be found in [44].

computational complexity and data association (i.e., the mapping between observations and landmarks in the map).

2.3 Topological SLAM approaches

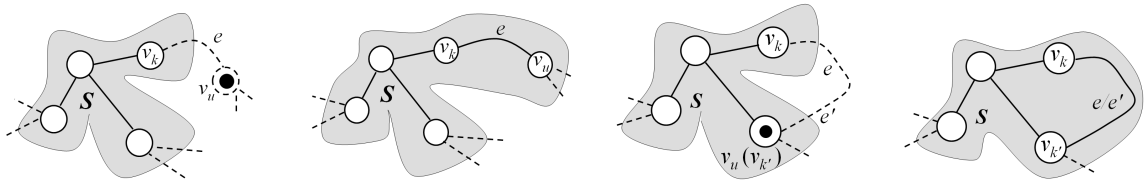
Topological approaches to SLAM adopt a topological representation that describes the connectivity of the environment. Without metric information, topological approaches typically rely on the use of marking aids to help the robot solve the loop closing problem deterministically. There also exist non-deterministic approaches where no marking aids or impoverished marking aids are used. Both approaches are reviewed below.

2.3.1 Deterministic topological SLAM approaches

The use of markers to help deal with state uncertainty was first explored in the 1960's and 1970's [48, 54, 11, 10]. In the literature, the 'markers' are also known as 'pebbles', 'tokens' or 'beacons'. Equipped with appropriate markers and minimal sensing and motion abilities, an explorer operating in a topological environment can solve the SLAM problem deterministically. One class of markers that has been examined extensively in the literature is a single movable marker that the robot can carry and use to mark one of the locations of the world uniquely. Another class of markers that has been examined extensively comprises sufficient distinct markers that the robot uses to uniquely label each visited location in the world. These algorithms are reviewed below, along with other marker classes.

Exploring with a movable marker

We start with marker-based approaches on undirected graphs. One notable work here is Dudek et al.'s single movable marker algorithm [24, 26]. In [24, 26] the world is modelled as a graph embedding consisting of a set of vertices and a set of edges between the vertices, and an ordering defined on the edges incident upon each vertex. This or a similar world model is also assumed in later work including [27, 23, 53, 22, 19, 17, 59, 60, 62] and is adopted in this thesis. Specifically, the world is defined in terms of an embedding of an undirected graph $G = (V, E)$ with a set of vertices $V = \{v_0, ..., v_{n-1}\}$ and a set of edges $E = \{(v_i, v_j)\}$. The graph is embedded within some space in order to permit relative directions to be defined on the edges incident upon a 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. Redundant or self-referential edges are prohibited in G. G is an *unlabelled* or *anonymous* graph as vertices and edges of G are not necessarily uniquely distinguishable to the robot. It is assumed that a robot can move from one vertex to another by traversing an edge. In [24, 26] the robot is equipped with an undirected movable marker that it can carry during exploration. The robot can pick up the marker if it is located at the current vertex and it can put down the marker it holds at the current vertex. The robot can identify when it arrives at a vertex. The sensory information that the robot acquires at a vertex consists of *marker-related* and *edge-related* perception. *Marker-related* perception encodes whether the marker is present at the current vertex, and *edge-related* perception encodes the relative orientations of edges incident on the current vertex in a consistent manner. No absolute distance and orientation information is available. Thus edges are featureless, and vertices



(a) Marker is not found in S. v_u is a new vertex

(b) Augment S by one vertex and one edge

(c) Marker is found in S. v_u is a known vertex

(d) Augment S by one edge

Figure 2.2: Single undirectional movable marker algorithm. S is augmented in (b) and (d). Dotted lines represent the unexplored portions of the graph-like world, and solid lines represent explored portion of the world. The position of the marker is represented by $\bullet$.

are featureless except for the exits to other vertices – two vertices appear identical to the robot if they have the same degree.

The SLAM algorithm developed by Dudek et al ([24, 26]) proceeds by incrementally building a known map S out of a known subgraph of the full graph. As new vertices and edges are encountered, they are added (represented) in S, and outgoing edges of new vertices are added to U which is the set of edges that lead to unknown places and thus must be explored. One step of the algorithm consists of selecting (and removing) an unexplored edge $e = (v_k, v_u)$ from U, and having the robot travel to the unknown end v_u, and answering the question 'have I been here before'? That is, disambiguating the unknown end vertex v_u and the entry edge e against previously explored ones (a.k.a. solving the loop closing problem). This problem is solved by the robot placing the marker at v_u and then visiting known vertices in S, looking for the marker. If the marker is not found at one of the vertices of S, then vertex v_u (where the marker has been dropped) is not in the explored subgraph, i.e., no new loop is formed (Figure 2.2(a)). In this case v_u is added to S. The previously unexplored edge e is also added to S becoming an explored edge, augmenting S by one edge and one vertex (Figure 2.2(b)). Other edges incident

on v_u are added to the unexplored edge set U. Alternatively, if the marker is found at some vertex $v_{k'}$ of the explored subgraph S, then vertex v_u (where the marker was dropped) is the known vertex $v_{k'}$ where the marker was found, i.e., a new loop is formed (Figure 2.2(c))). In this case, 'back-link validation' is required, i.e., inferring the incident edge e' at $v_{k'}$ that edge e correspond to – 'by which exit (edge) of $v_{k'}$ did I entered $v_{k'}$?' To solve this, the robot drops the marker at v_k and goes back to $v_{k'}$ along the shortest known path in S. At $v_{k'}$, the robot traverses each of the (unexplored) incident edges at $v_{k'}$, looking for the marker. One of the (unexplored) edges e' will take the robot back to v_k, which the robot will immediately recognize due to the presence of the marker. Edge $e/e' = (v_k, v_{k'})$ is then added to S and removed from U. In this case S is augmented by an edge (Figure 2.2(d)). After the disambiguation, the algorithm proceeds to the next iteration in which the above steps are repeated with a newly selected edge from U. The algorithm terminates when the set of unexplored edges U is empty, with map S being isomorphic[2] to the world G. Assuming one mechanical step for the traversal of one edge, the main cost of exploring the graph G in terms of edges traversed by the robot (mechanical complexity) is $O(mn) \leq O(n^3)$, where m and n are the number of edges and vertices in G [26].

The model and algorithm developed in [26] has inspired work by the same authors and others. Closely related work includes [25, 27, 19, 17, 53, 23, 28, 60, 61, 62]. Work by the same authors [27] assumes the same world model as in [26] and investigates two problems that are related to the exploration problem discussed in [26], namely the 'Map Validation' and 'Self-location' problems. In the Map Validation problem, the robot is given a map of a graph-like environment, which is of

[2]Following the extended graph isomorphism as defined in [24].

the same form as the one computed by using the exploration algorithm in [26]. The robot is told which map vertex is its current location, as well as the correspondence between one map edge incident on the current map vertex and a physical exit from the current physical vertex. That is, the position and orientation of the robot with respect to the map is known. The problem is to verify the correctness of the map, i.e., to determine whether the map is consistent with the world by looking for an isomorphism relationship between the map and the world. The key idea underlying the validation algorithm is to construct a spanning tree rooted at the current vertex. The algorithm first verifies the presence of this tree in the world, and then verifies the remaining edges of the graph-like world. Similar to the exploration task, the validation process uses the marker for location disambiguation. During validation, whenever the information that the robot senses about the real world does not match the expected information modelled by the map, the validation fails. The sensed information includes the degree of the node visited, and the presence or absence of a marker at the node. This sensory information defines the signature [40] of the vertex, which allows it to be locally distinctive within its immediate neighborhood. The validation algorithm requires $O(n^2)$ moves (edge traversals) in the worst case. The paper then investigates the Self-location problem, which is a more general problem in which the robot is given a map of its environment to be verified, but is not told its location and orientation with respect to the map. The paper gives an algorithm for the self-location problem that uses $O(n^3)$ moves. The idea behind the self-location algorithm is first to form all possible hypotheses using the given map, corresponding to all possible initial vertices and orientations (i.e., their reference edges) in the map, and then to explore the graph using the marker, discarding hypotheses which are

found to lead to inconsistencies during exploration. That is, whenever the information the robot senses about the real world (marker and degree information) does not match the information modelled by a specific hypothesis, that hypothesis is rejected. When the exploration process is complete, either no hypothesis remains, or, one or more hypotheses remain. In the former case no starting pose was consistent with the world, and the map must be incorrect. In the latter case no inconsistency is observed between the hypothesized initial pose(s), the map, and the true starting pose and true environment, thus the map can be used for navigation and path planning, and any one of the starting pose(s) can be assumed to be correct.

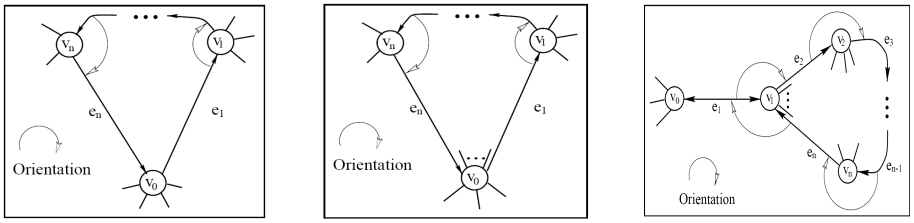
Deng et al. [19] also follows the world model introduced by [26] and conducted a competitive analysis for the performance of different strategies, including the exploration algorithm in [26]. The main idea of competitive analysis is to evaluate how good a strategy that operates under incomplete information is by comparing it against the optimal solution with complete information. In this approach, as in the authors' earlier work [20], exploration strategies are evaluated by examining the (worst case) ratio of the cost of building the map (where the robot initially knows nothing about the world) to the cost of verifying the map (where the robot has a map of the world and knows its initial position and orientation in the map, but still wants to verify the correctness of the given information). The *competitive ratio* of a strategy is defined as the maximum ratio, over all allowable graphs, of the number of traversed edges for establishing the map to the minimum number of edges traversed for verifying a map of the same graph. A mapping strategy with the competitive ratio c always traverses a total number of edges which is no more than c times the number of edges traversed in verifying the map. An algorithm that minimizes the ratio

is considered the optimal algorithm. The paper shows that the single movable marker algorithm by Dudek et al. [26] is of competitive ratio $O(n)$. The paper also shows that for the model in [26] and the single marker case, the result of [26] is asymptotically optimal within a fairly reasonably restricted class of strategies (i.e., Depth-One search strategies which always drop a marker at the unknown end of an edge and comes back). The paper shows this by constructing a special subclass of embedded graphs called star-shaped graphs which has ratio $\Omega(n)$. This ratio is shown to be a lower bound over all embedded graphs for depth-one strategies, thus establishing a tight bound of competitive ratio $\Theta(n)$. The paper also shows that the competitive ratio of depth-one strategies for mapping embedded *planar* graphs with a single movable marker is $\Omega(\log n)$.

Given the high potential cost associated with single robot exploration and mapping, Dudek et al. [28] extended the concept of a single robot exploring a graph-like world to the case of multiple robots. [28] sketched how multiple robots might exploit the algorithms developed in [24, 26] to explore in a coordinated fashion. A critical assumption of [28] was that the individual members of the robot team were limited to communication when they were in the same graph node, and thus multiple robot exploration requires coordinated exploration and map merging in order to be effective. In [59], I formally developed the approach suggested in [28] and evaluated the performance of the two robot exploration algorithm relative to that of a single robot exploring the same environment. Both [28] and [59] assume the same environmental representation as described in [24, 26] and populate the world with two or more robots, each of which is equipped with its own undirected movable marker. Joint exploration is achieved through alternating phases of independent exploration by the individual robots and coordinated merging of the independently

acquired partial world representations. The markers are used for disambiguation purposes in both the independent exploration and merging phases. Empirical evaluation conducted in [61] showed that exploration with two robots can provide improvements in exploration effort required over that of a single robot, and that for some environments this improvement is super-linear in the size of the graph. Some enhancements to the basic multiple exploration algorithm in [28, 59] were also investigated in my later work [60, 61, 62].

Variations of the undirected graph-like world model also exist. For example, in [53] Rekleitis and Dudek present a deterministic algorithm for exploring an undirected *planar* graph with a movable marker. This constraint permits mapping and exploration to take place much more efficiently than required by the more general algorithm described in [26]. By exploiting the fact that a connected planar graph can be decomposed into a set of cycles, which are called *'ears'* in the paper, [53] develops an exploration algorithm for exploring planar graphs with cost that is typically linear in the size m of the graph. In contrast to the technique described in [26] where a single vertex is added after validation, here a closed path (ear) is added after validation. The single movable marker is used to mark the starting vertex of the explored ear. Specifically, at a selected vertex where there is an unexplored edge the robot drops the marker in order to mark the starting vertex and then starts the exploration of the ear by making only "right turns" until it returns to the marked vertex, after, say, p edge traversals. Now the robot needs to determine the 'relationship' of the incoming (entry) edge and the outgoing edge by which it started the exploration, which is required for connecting the newly explored ear to the explored subgraph. The robot picks up the marker and backtracks to the previous vertex visited where it drops the



(a) Empty ear (b) Non-empty ear (c) Isthmus

Figure 2.3: Three cases in exploring ears. Courtesy of Dudek et al. [53].

marker, then the robot continues backtracking until it either reaches the marked vertex or it has performed p edge traversals. There are three different cases depending on the topology of the explored graph: 1) *Empty Ear*: After an additional p traversals the robot finds the marker. In this case the robot has explored an empty ear and the incoming edge is adjacent to the outgoing edge (Figure 2.3(a)). There are no nodes inside such an ear as at every node the immediate neighbor was selected. 2) *Non Empty Ear*: After an additional p traversals the robot has not found the marker. In this case the robot has explored an non-empty ear (Figure 2.3(b)). In order to reveal the number of edges between the outgoing edge and the incoming edge, a sequential one-step search of the marker is conducted – by traversing every edge adjacent to the current entry edge. 3) *Isthmus*: The marker is found after an additional $p - 2$ steps. In this case the incoming edge is identical to the outgoing edge (Figure 2.3(c)). The explored path encloses a subgraph that has a single connection to the rest of the graph via the starting vertex. The explored subgraph is then updated using a recursive procedure which merges the newly explored vertices with the existing ones. The authors explain that the efficiency of their approach derives from the fact that it is specific to planar graphs only.

In the work reviewed above, a single *undirected* movable marker is used in exploring an undirected graph-like world. There also exist results on exploring undirected graphs using a *directional* movable marker. As pointed out in [27], a directional movable marker is in general more powerful than an undirected movable marker. [27] enhances the single undirected movable marker algorithm in [26] using a directional movable marker. With a directional movable marker, the robot can uniquely identify a (marked) node and a local direction (an edge) in the vertex. Whenever the robot finds the marker at a known vertex $v_{k'}$, by enumerating the edges and identifying the one that is pointed to by the marker, the robot can trivially infer the ordering (label) of the edge e' of $v_{k'}$ that corresponds to e. Using a directional marker thus avoids the need for the 'back-link validation' process in the original algorithm. With a directional movable marker, the exploration cost is still $O(mn)$ but the constant is expected to be reduced. Deng et al. [17] also adopts the same graph representation introduced in [26] and investigates the map validation problem using a movable directional marker that can be put down on an edge of the graph world G and picked up later as needed. In the validation problem the robot is given a map M of the world G with its position and orientation indicated on the map, the task is to find out whether map M is correct for the world G. The map M is a *plane* embedded graph (even though the world model G may or may not be planar). The general idea of the algorithm is to trace the *faces* [36] of the augmented map M one by one, mimic the action on the environment graph, and compare the local observations (the degree of the node visited and the presence or absence of a marker at the current node or edge) during tracing against the information on the map. The paper develops a strategy that verifies the map by traversing each edge at most 4

times. The algorithm does not work when the map cannot be embedded in a plane.

There also exist marker-based papers that model the world as a strongly-connected *directed* graph (e.g., [8, 7]). Exploring a directed graph using a movable marker is more difficult than exploring an undirected graph. In contrast to the case of an undirected graph, on a directed graph the robot may not be able to retrace its steps to retrieve the marker or explore. [8] argues that exploring a directed graph with two robots is more powerful than one robot with multiple markers. The authors argue that a single robot with a constant number of movable markers cannot efficiently explore strongly-connected directed graphs without *a priori* knowledge of the number of vertices n. The authors conjecture that the same holds when n is known. The results of [8] motivate two questions: (1) How many undirected movable markers are required in order to learn directed graphs efficiently if n is known? (2) How many such markers are in fact needed if n is unknown? Later work presented in [7] demonstrates that surprisingly few markers are needed in both cases. [7] shows that (1) If the robot knows n (or an upper bound $\widehat{n}$ on n), it can learn the graph with only one undirected movable marker in time polynomial in n (respectively, $\widehat{n}$). (2) If the robot does not know n (or $\widehat{n}$), then $\Theta(\log \log n)$ such markers are both necessary and sufficient. The results disprove the conjecture given in [8] that one robot with a constant number of markers cannot (efficiently) map a directed graph even when upper bound on the number n is known. Whether a single directional movable marker is sufficient to solve the mapping problem in directed graphs (without prior knowledge of n) remains an open question.

Exploring with 'sufficient distinctive' markers

The work reviewed above addresses exploration and mapping on unlabelled graph-like worlds. Another deterministic mapping problem that has been investigated extensively involves exploring on *labelled* graphs [45, 46, 21]. In such work, either all the vertices and edges in the graph contain distinct labels, or the graph is unlabelled but the robot has sufficient distinct markers such that it can uniquely label each visited vertex and edge as it explores, and can recognize the labels when encountered again. In both cases, the robot can recognize already visited vertices and traversed edges. Such a 'strong' marking aid can be implemented, for example, with $m + n$ distinct (immovable) markers, one on each visited vertex and edge, or, $\binom{m}{2} + \binom{n}{2}$ homogeneous markers that are used to simulate distinct markers (by using i identical markers in place of a distinct marker with label i)[3]. By marking each visited vertex and edge uniquely, at a new vertex the robot can determine 'have I been here' and 'by which exit did I arrive' without any further validation. Without the need for both validations, the mapping problem can be solved using search algorithms such as Depth-first search (DFS) and GREEDY, which have $O(m)$ cost. Recent work in this paradigm either deals with evaluating and improving DFS, GREEDY and other existing search algorithms, or addresses more challenging problems such as constrained exploration. Some of this work is reviewed here.

In evaluating existing searching algorithms and developing new ones, [45] defines the *penalty* of an exploration algorithm running on a graph $G = (V, E)$ to be the worst-case number of traversals in excess of the lower bound $m = |E|$ that the robot must perform in order to explore

[3]Such a marking aid can be implemented in other ways which involve fewer markers, as discussed in Chapter 6.

the entire graph. (The lower bound m can be achieved in the 'best case' – for Eulerian graphs, by an off-line algorithm provided with a labelled map of the graph with known starting point as well as the other ends of all edges incident on the currently visited node.) The aim of [45] is to give an exploration algorithm with penalty linear in the number of nodes n. The paper first shows that two natural exploration heuristics, the GREEDY and the depth-first search (DFS) algorithms cannot achieve this efficiency. in the graph. These two algorithms give a $\Theta(m)$ penalty, which may become quite large in the case of dense graphs. By maintaining a dynamically constructed tree structure which interconnects saturated vertices – vertices having no more unexplored edges – and using the tree to control the traversals on explored edges, this paper then gives an exploration algorithm with penalty that never exceeds $3n$.

Under the same graph and marker model, later work by the same group of authors [46, 21] investigates the impact of the amount of *a priori* topographic information available to an exploration algorithm on its performance. In [46] the authors consider three cases, providing the robot with varying amount of *a priori* information. The robot may either: (1) have a complete *a priori* knowledge of the graph – a labelled map of the graph along with a '*sense of direction*'. Having 'sense of direction' means that when arriving at any node v, the robot knows the label of v as well as the label of the other end of every edge incident to v. (2) have only an un-oriented map of the graph, i.e., an unlabelled isomorphic copy of the graph and cannot locate its position on the map and, when arriving at a node, does not have any *a priori* knowledge concerning the other ends of yet unexplored edges incident to it. (3) finally, lack any knowledge of the graph. The paper studies the impact of the varying amount of knowledge on the exploration performance

using an approach similar to competitive analysis, i.e., considering the ratio of the cost of an algorithm lacking some knowledge of the graph to that of the optimal algorithm having this knowledge. For a given graph, both costs are maximized over all starting nodes and the ratio of these maxima is considered as the performance measure of the algorithm on the graph. It is shown in [46] that the best exploration algorithm lacking any knowledge of the graph (case 3) requires twice as many edge traversals in the worst case as does the best algorithm which has an un-oriented map of the graph (case 2), i.e., the ratio is two. The DFS algorithm is one such optimal algorithm, as it does not rely on any *a priori* knowledge of the graph and its cost does not exceed twice the number of edges. On the other hand, the latter (case 2) uses twice as many edge traversals in the worst case as does the best algorithm having complete knowledge of the graph (case 1).

Closely related work [21] considers three similar but slightly different scenarios with different amounts of information. (1) The robot has an unlabelled isomorphic copy of the explored graph with a marked starting node. This is called an *anchored map* of the graph. (2) The robot has an unlabelled isomorphic copy of the explored graph. This is called an *unanchored map* of the graph. (3) The robot lacks *a priori* knowledge of the explored graph. Note that even the scenario with an anchored map (case 1) does not give the robot any 'sense of direction', since the map is unlabelled (although the graph is labelled or can be labelled during exploration). For example, when the robot starts the exploration of a line with an anchored map, such a map gives information about the length of the line and distances from the starting node to both ends, but does not tell which way is the closest end. This paper uses the notion of *overhead* which is similar in spirit to the cost

<table>
<tr><th></th><th>Anchored Map</th><th>Unanchored Map</th><th>No map</th></tr>
<tr><td>Lines</td><td>overhead: 7/5
optimal</td><td>overhead $\sqrt{3}$
optimal</td><td rowspan="3">DFS
overhead 2
optimal</td></tr>
<tr><td>Trees</td><td>overhead: 3/2
optimal</td><td>overhead: < 2
lower bound $\sqrt{3}$</td></tr>
<tr><td>General graphs</td><td colspan="2">DFS, overhead 2, optimal</td></tr>
</table>

Table 2.1: Summary of results in [21].

measure described in [46] as a measure of the quality of an exploration algorithm. The overhead of an algorithm is the ratio of its cost to that of the optimal algorithm having full knowledge of the graph, maximized over all starting nodes and over all graphs in a given class. Using overhead as a measure, the paper presents exploration algorithms that have optimal overhead for all of the above scenarios, except case 2 for trees. The paper shows that while for the class of all undirected and connected graphs, DFS turns out to be an optimal algorithm for all scenarios, the situation for trees and lines is much different. Specifically, under the scenario lacking any knowledge (case 3), DFS is still optimal for trees and lines but this is not the case if a map is available. Under the scenario of an anchored map (case 1), the paper constructs optimal algorithms for trees and lines with overheads of 3/2 and 7/5 respectively. For the scenario of an unanchored map (case 2), the paper shows that for lines the optimal overhead is $\sqrt{3}$ and for trees the optimal overhead is at least $\sqrt{3}$ but strictly below 2. (Thus DFS, which has overhead 2, is not optimal for trees and lines.) The paper constructs an algorithm for trees with overhead ≤ 1.99. The paper concludes that the construction of an optimal exploration algorithm with an unanchored map for the class of trees, and establishing the value of the best overhead remains an open problem. A summary of the results is given in Table 2.1.

There exists some work that assumes a sufficient supply of distinctive markers in exploring

an undirected graph but adds the constraint that the robot has to return to the starting point periodically (say, for refueling). This problem is termed *piecemeal exploration* of an undirected graph [9, 4]. Later work [30] investigates a related but perhaps more constrained case – *tethered exploration*. In tethered exploration the robot is tied to the starting node by a tether (rope). If the tether has length l, then the robot must remain within distance l from the starting node s. (In practical terms the rope can be a fuel line, or a communication line, or a safety line.) Although the tethered robot is not constrained to return to s periodically as in piecemeal exploration, it might be forced to backtrack (rewind the rope) a great deal just to visit an adjacent vertex. In both piecemeal exploration [9, 4] and tethered exploration [30], the world is modelled as a finite connected undirected graph $G = (V, E)$ with a distinguished start vertex s. It is assumed that the explorer can always recognize a previously visited vertex and that it never confuses distinct locations. At any vertex the robot can distinguish between incident edges at any vertex. (Each edge has a label that distinguishes it from any other edge.) At a vertex, the robot knows which edges it has traversed already. The explorer's goal is to learn a complete map of the environment (graph) – by visiting every vertex and traverse every edge, while minimizing the total number of edges traversed.

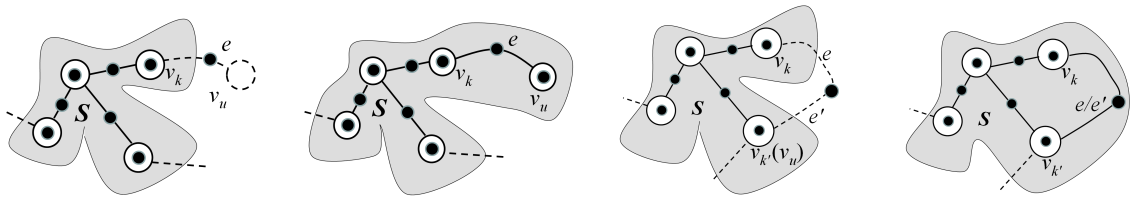
We conclude this review section on exploring labelled graphs by surveying additional results on exploring *directed* graphs, which are labelled or can be labelled by the robot during exploration. Work in this paradigm includes [20, 1, 42, 33]. Deng et al. [20] show that the Eulerian property is central in this problem. The main contribution in [20] is that it demonstrates that the graph exploration problem for graphs that are very similar to Eulerian graphs can be solved efficiently.

They use a parameterization called *deficiency* to express how similar a graph is to an Eulerian graph. Deficiency is the minimum number of edges that must be added to make a graph Eulerian. The paper shows that if the graphs have deficiency one and the deficiency is known *a priori*, then there exists a strategy that never traverses an edge more than four times. The paper also presents a generalized algorithm for directed graph of deficiency d, which never traverses an edge more than $d^{O(d)}$ times, i.e., the algorithm achieves an upper bound of $d^{O(d)}m$. Albers and Henzinger [1] gave a first improvement to the algorithm in [20]. They presented a sub-exponential *Balance* algorithm which can explore a directed graph of deficiency d with upper bound of $d^{O(\log(d))}m$. They also claimed that this bound was tight for their algorithm by showing a matching lower bound of $d^{\Omega(\log(d))}m$. The authors also gave lower bounds of $2^{\Omega(d)}m$ edge traversals for several natural exploration algorithms including Greedy, Depth-First, and Breadth-First. Since this work, there have been additional results in determining whether a graph with deficiency d can be explored by traversing $O(poly(d)m)$ edges. [42] develops a depth-first search algorithm that obtains the bound when the graph is dense. More recent work [33] gives the first generalized polynomial d algorithm. This paper proves that the algorithm needs at most $O(d^8m)$ edge traversals. The authors conjecture that this bound can probably be improved.

Exploring with multiple homogeneous markers

There are results on exploring graphs with multiple homogeneous (indistinguishable) markers. For example, [7] shows that there exists a deterministic algorithm that can map a directed graph in polynomial time using $O(\log \log n)$ movable markers. There also exist results on exploring

with multiple homogeneous markers that are *immovable*. Deng and Mirzaian described in [18] a deterministic algorithm for a robot to map an undirected graph using the *footprints* model. For the mapping problem in the footprint model, the robot has the power of knowing whether a node or an edge has been visited before, though it may not remember when and where it was visited. The footprint model is reducible to the marker model with $m + n$ homogeneous immovable (undirected) markers, one for each node or edge. The robot drops one marker at each vertex and edge the first time the vertex or edge is visited, and does not pick it up again. That is, the robot leaves unerasable 'toe-less footprints' on the underlying graph during exploration. For each such location, the footprint answers the question 'have I been here before?', but it does not tell which visited location the current location is. For ease of exposition, we describe the algorithm following the sketch of the single marker algorithms described in [26]. The footprints algorithm maintains a (foot-printed) known subgraph S and an unexplored edge set U. One step of the algorithm consists of selecting an unexplored edge e in U and traversing from the known end vertex v_k to the unknown end v_u via e (and leaving a footprint/marker on e). If v_u contains no marker then it must not have been visited before (Figure 2.4(a)). The robot leaves a footprint/marker at v_u. Both e and v_u (now foot-printed) are then added into S without additional validations (Figure 2.4(b)). If v_u contains a footprint/marker already then it has been visited before (Figure 2.4(c)). Now the robot needs to determine which known vertex it is visiting, and by which edge it entered the vertex. The newly dropped footprint/marker on e is exploited in identifying both v_u and the edge e' leaving v_u which correspond to e. The newly footprinted edge e results in vertex v_u and v_k each having one additional footprinted edge. Validations are



(a) v_u was not visited before

(b) Augments S by e and v_u

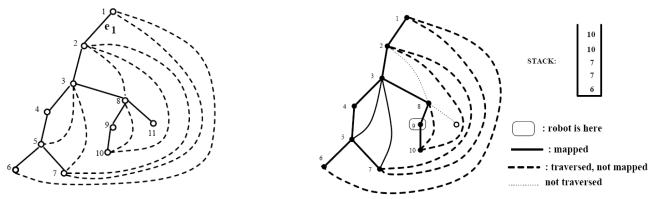
(c) v_u was visited before

(d) Augments S by e/e'

Figure 2.4: Mapping with undirected homogeneous markers (footprints). Markers/footprints are represented by •.

conducted by the robot returning to v_k and then visiting each (other) known vertex in S which has the same degree as v_u, looking for the (other) vertex that has one additional footprinted edge. Call this vertex $v_{k'}$ (which corresponds to v_u). Once $v_{k'}$ is identified, this 'unexpected foot-printed edge' $e/e' = (v_k, v_{k'})$ is added to S (Figure 2.4(d)). While the exploration cost is $O(mn)$ as the robot may have to exhaust all the vertices in S for validating a single edge, the algorithm is expected to produce a reduced cost over the single marker algorithms, due to the reduced need for validation-related motions.

Within the same 'footprints' model, Deng and Mirzaian also described in [18, 19] an algorithm for a robot to map an undirected *planar* graph by traversing each edge a constant number of times. The backbone of the algorithm is a *rightmost depth-first search* which generates a DFS-tree. The starting node of the search becomes the root of the DFS-tree, and the first edge leads to the leftmost child of the root. From then on, whenever the robot has to select the next edge out of the current node, in order to continue the DFS, it always selects the rightmost one available, i.e., counter-clockwise first. Such a traversal of the graph gives a DFS-tree which the authors refer to as *rightmost DFS-tree*. The non-tree edges are called back-edges. The crucial property



(a) (b)

Figure 2.5: A rightmost-DFS traversal of a planar embedded graph. Courtesy of Deng et al. [19].

of a rightmost DFS-tree is that all the back-edges appear on the right 'shoulder' of the tree (Figure 2.5(a)). This forces a generalized nesting (or parentheses) structure among the back-edges. This nesting property is exploited in the algorithm. By using stacks, the paper develops a footprint algorithm that maps an unknown embedded planar graph by traversing each edge at most twice. [18, 19] also shows that with some modifications, the method can be applied for exploring such embedded planar graphs with only n homogeneous markers which are placed on nodes but not edges, i.e., having footprints at nodes but not on edges. The modified algorithm uses extra traversals to compensate for the absence of footprints (markers) on traversed edges, at the cost of increasing edge traversals for each edge from two to four.

Exploring with other marking aids

Finally, it is worth mentioning that within the topological exploration literature there exists work that assumes marker classes that are more powerful than the 'sufficient distinctive' markers that can label the graph uniquely. For example, some work augments the robot with a movable marker that can have messages left on it (e.g., [12]), while other work associates each vertex with

a *whiteboard* on which the robot writes and reads messages (e.g., [6]). Some work combines more than one such marker class (e.g., [12]). Usually, a more powerful marking aid is used to address more challenging problems and usually involves multiple robots. Some of this work is briefly introduced here. [15, 6] address the distributed version of the graph exploration problem. Here there are k identical robots initially dispersed among the n nodes of the graph. The objective is for each robot to build a map of the graph such that each map is consistent with one another in terms of the vertex labelling, i.e., the label assigned to any node should be the same in every robot's map. The exploration is conducted in a distributed fashion, and a whiteboard is associated with each vertex. This whiteboard is used both for marking the vertex, and more importantly, for providing 'indirect' communication among the robots. (As in the movable marker-based multiple robot algorithm described earlier, it is assumed here that the robots cannot communicate when they are not in the same vertex.) In this work the whiteboards are used in both the distributed exploration and merging phases. Other work including [34, 12] uses whiteboards combined with other marking aids to address the distributed exploration problem in 'dangerous' environments. The environment graph representation in which the agents operates is dangerous due to the presence of 'harmful' nodes and edges, called *black holes* and *black links*, which destroy any incoming robot without leaving an observable trace. Due to this danger, multiple (distinct) robots are usually required to ensure that at least one will survive and finish the task. The problem is for the team of agents to explore the world and, within finite time, to construct a map of all the safe nodes and edges and indicate the locations of all the black holes and black links. The problem is considered solved if at least one agent survives, and all surviving agents terminate

within finite time with such a map generated. [34] uses both fixed and movable whiteboards. The movable whiteboards are termed 'markers' in the paper but the markers can have messages written on them. Fixed whiteboards are associated with each vertex, and movable whiteboards are initially associated with each starting location of the robots and can be carried by the robots.

Summary and open problems for marker-based topological approaches

This section reviewed notable work on robotic exploration and mapping using a topological representation. In general it is assumed that the robot has very limited metric measurement, which is insufficient for the robot to distinguish locations alone. A marking aid is used by the robot to solve the 'have I been here before' problem. One instance of the problem that has been examined extensively is that of a single movable marker that the robot can carry and use to mark one of the locations of the world at a time. A second instance of the problem that has also been examined extensively comprises sufficiently many distinctive markers that the robot can use the set of markers to uniquely label each visited location in the world. There are also results in exploring with multiple homogeneous markers and markers that contain messages. While most of the work models the environment as an undirected graph, there also exist results on exploring directed graphs. The marker-based work reviewed in this section is summarized in Table 2.2, which also summarizes the non-deterministic exploration strategies reviewed in the next section. Known solvability results and cost bounds of topological exploration on undirected graphs with different marker classes are summarized in Table 2.3.

There are a number of open problems within the topological formalism that have not been

Summary of topological approaches reviewed in Section 2.3		
Marker classes	Undirected graph	Directed graph
A movable marker	Dudek et al. [24, 26, 28, 27] Hui et al. [60, 61, 62] – (*multi-robot*) Rekleitis et al. [53] – (*planar graph*) Deng et al. [19, 17] – (*edge marker*[17])	Bender & Slonim [8] Bender et al. [7]
$m+n$ homogeneous markers (Footprints)	Deng et al. [18, 19]	
$m+n$ distinct markers (labelled graph)	Panaite & Pelc [45] Pelc et al. [46, 21] Awerbuch et al. [9, 4] – (*piecemeal*) Duncan et al. [30] – (*tethered*)	Deng et al. [20] Albers & Henzinger [1] Kwek [42] Feischer & Trippen [33]
More powerful marking aids	Das, Barriere et al. [15, 6] – (*whiteboards*) Flocchini et al. [34, 12] – (*movable whiteboards*)	
Non-deterministic topological approaches	Dudek et al. [22, 23, 29, 43] – (*marker-less*) Werner et al. [69] – (*insufficient label*) Tully et al. [58] – (*probabilistic*)	

Table 2.2: Topological exploration algorithms reviewed in this chapter.

fully addressed in the literature. These include: Investigating the relative powers of different classes of immovable markers; Determining if a single immovable marker can be used to solve the SLAM problem deterministically, as is known to be the case for a movable marker; Identifying efficiencies that exist in different immovable marker classes; And, identifying what advantages exist in the use of other types of markers, such as multiple immovable markers and strings.

Marker classes		Solvability and upper bounds
A single movable marker	undirected	yes. $O(mn)$
	directional	yes. $O(mn)$
$m+n$ homogeneous markers (footprints)	undirected	yes. $O(mn)$
	directional	yes. upper bound not known
$m+n$ distinct markers (labelled graph)	undirected	yes. $O(m)$
	directional	yes. $O(m)$

Table 2.3: Solvability and known cost of topological exploration of the different marker classes reviewed in Section 2.3.1 for undirected graphs. Note that the trivial lower bound for the topological exploration and mapping problem is $\Omega(m)$.

2.3.2 Non-deterministic topological SLAM approaches

Using additional marking aids such as a single movable marker or multiple markers that are sufficiently distinctive, a graph-like world can be fully explored and mapped deterministically, even if there are no spatial metrics and little sensory ability on the part of the robot. These approaches are provably correct deterministic solutions to the SLAM problem. Without a marker, or with insufficient markers, the mapping may not be solved deterministically. Some of the non-deterministic approaches are reviewed below.

Exploring without marking aids

Under the same world and robot sensing model as assumed in [26], Dudek et al. investigated the possibility of exploring a topological environment *without* any marker [22, 23]. These papers present an exploration approach simply based on the structure of the world itself. In contrast to the above deterministic marker-based approaches where a single unique solution is produced, here multiple models of the unknown world may result. During exploration the robot incremen-

tally constructs an *exploration tree*, which includes, at the end of the exploration, the set S of all possible world models that are consistent with the robot's observations. The *nodes* of the exploration tree represent possible partial models of the world. *Leaf* nodes represent possible models (complete configurations) of world connectivity and are the elements of S. A given node in the exploration tree is considered to be a leaf node (i.e., a possible model) if there are no paths still to be traversed. Solving the 'have I been here before' problem here is challenging since place identification must be performed with very limited information. Indeed, by associating the signature of a place with the vertex degree (only), the robot cannot always know whether it is visiting a place for the first time or not. Thus, when the robot visits a place it must consider *all* possible ways of adding vertices to the frontier nodes in the exploration tree. Three classes of errors or mis-identifications can be identified when the robot visits a given vertex v_i.

1. Errors of type OLD-LOOKS-NEW. Vertex v_i is assumed to be a new vertex even though it has been visited before (failure in correspondence). In this case, an additional vertex is added to represent the current place even though a vertex for the current place has already been created.

2. Errors of type NEW-LOOKS-OLD. Vertex v_i is assumed to be a previously visited vertex even though it is new. In this case, the map will have a missing vertex relative to the real world and incorrect connectivity.

3. Errors of type MIS-CORRESPONDENCE. Vertex v_i is 'recognized' as a known vertex v_j ($j \neq i$) even though, in reality, it is another old vertex v_k (i.e., the robot has confused two known nodes); Thus, an erroneous edge is added to the world.

Branches in the exploration tree are created as a result of modelling the true topological structure of the world, or by making one or more correspondence errors of different types. The development of any branch is halted once the frontier node has no more paths to traverse. Note that the exploration tree will always contain a branch which leads to a leaf describing the real world, where no errors are committed. See Figure 2.6 for an example of exploration tree and different types of errors. In Figure 2.6, the example solution universe S consists of two leaf nodes (denoted $M7$ and $M8$ in the figure), with leaf $M8$ being the correct model. To make the exploration more robust and effective, the algorithm exploits non-local information by defining an *extended signature* that incorporates signature information about a vertex's neighbors. Despite the availability of an extended signature, ambiguity may still exist in place identification. As a result, the universe of possible solutions S may contain various models which are equivalent insofar as the extended signature is concerned, of which just one faithfully reflects the connectivity in the world. For example, in single cycle graphs every place is identical to every other and the number of possible models grows infinitely, as shown in Figure 2.7. This difficulty is not surprising since under such circumstances the algorithm is attempting to construct a map without knowledge about where the robot is or how the robot is moving, and all nodes may have identical signatures. The paper suggests using some additional information about the environment as a cue to cease exploration even though some possible models have not been fully explored. An example cue is simply the prior knowledge of the number of locations (vertices) n in the world. By exploiting this cue the solution set can be reduced, since now the exploration process can terminate as soon as all models in the exploration tree have n vertices.

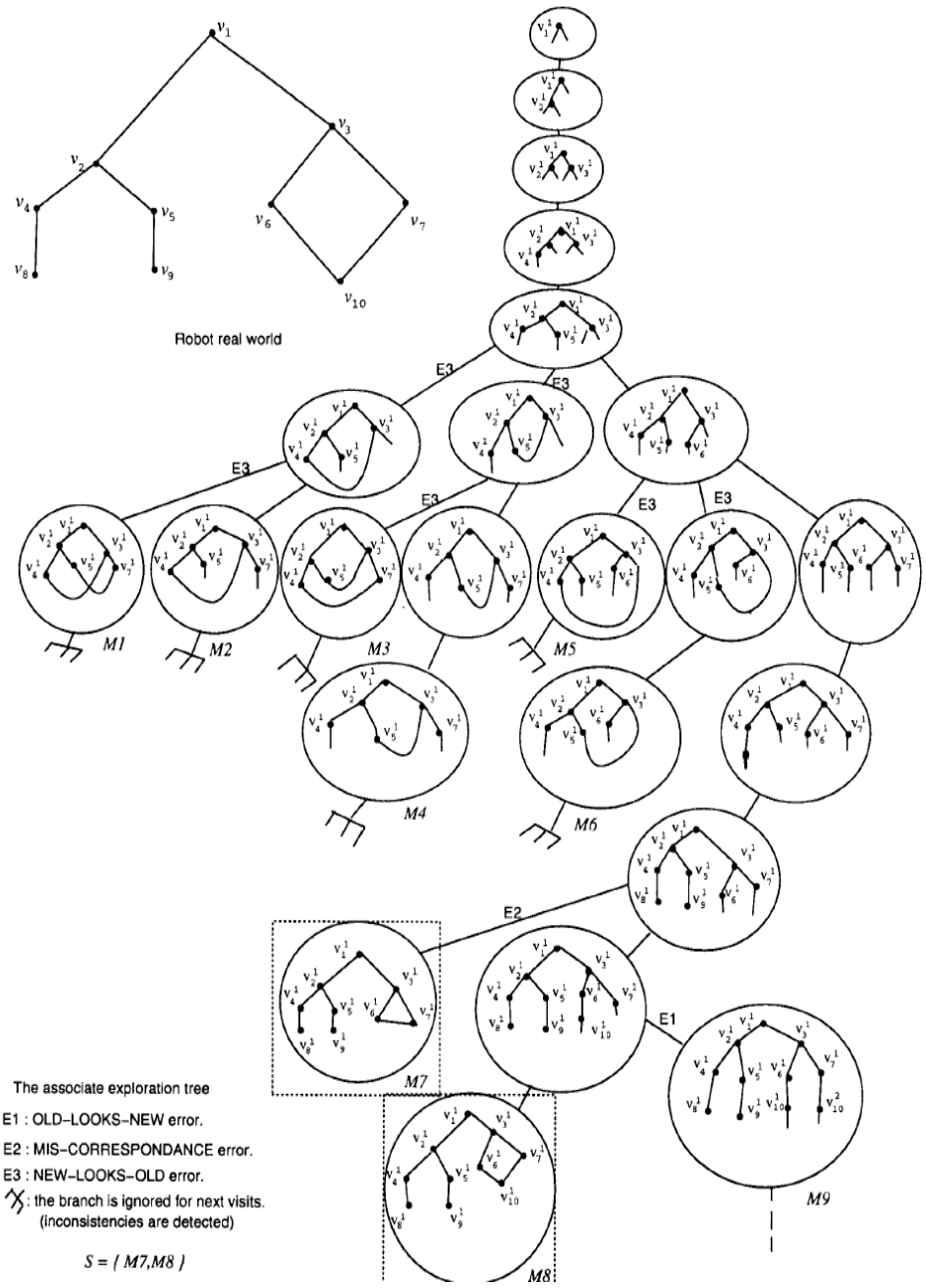


Figure 2.6: Exploration tree and three types of errors. Courtesy of Dudek et al. [22].

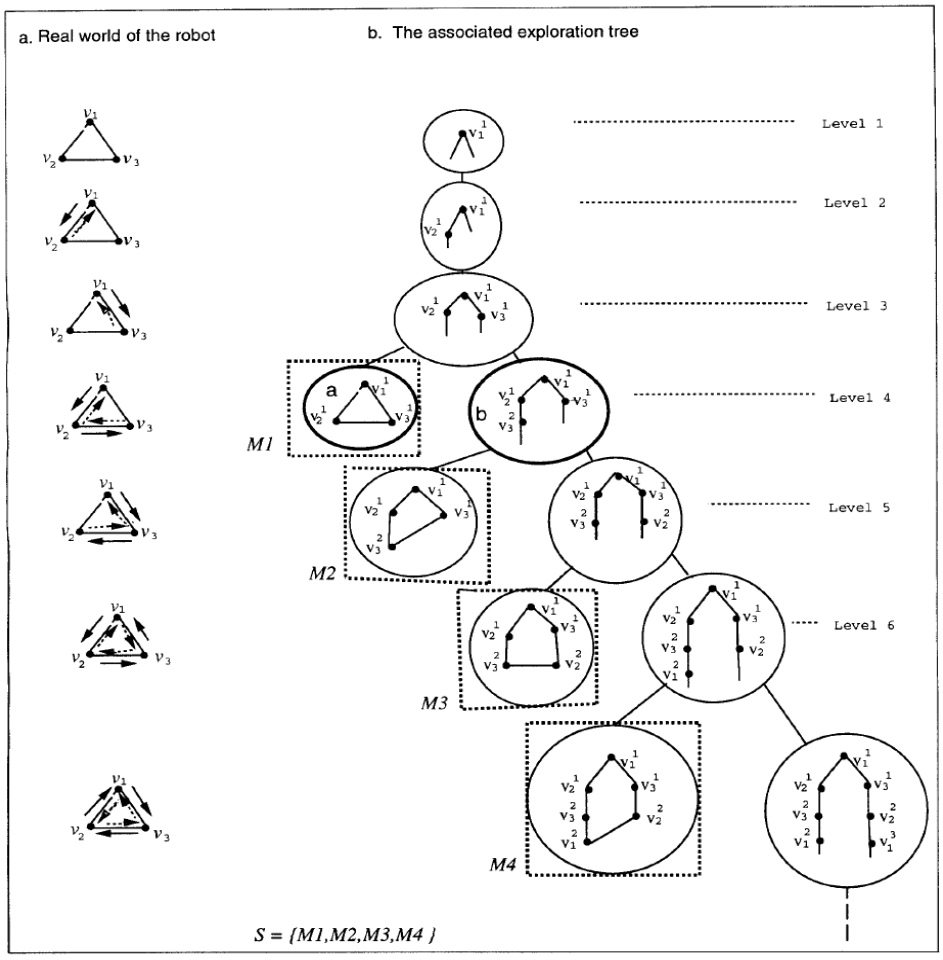


Figure 2.7: Problems with constructing the exploration tree for a regular graph. Courtesy of Dudek et al. [22].

The marker-less approach in [22, 23] was revisited by Dudek et al. in more recent work [29, 43]. The paper presents a new exploration strategy called *breadth-first ears traversal* (BFET) that can be used on embedded *planar* graphs. BFET works by eliminating inconsistent models through re-visiting of previously explored vertices in a cyclic manner. The paper also presents a stochastic variant of BFET that attempts to capture the spirit of this approach. The paper also describes a search algorithm which bounds the number of hypotheses maintained at each step of

the exploration process based on a heuristic evaluation function, which, following Occam's razor principle, considers the simplest model capable of explaining the observed data to be the best one. Accordingly, at each traversal of an edge during the exploration process the algorithm first enumerates the new models that can be generated from each of the currently maintained world hypotheses, and then ranks them using the heuristic function – based on the number of vertices in the model. The top N of these models are then selected for maintenance and the rest are discarded. This approach is conceptually similar to the use of particle filters in metric SLAM algorithms, although instead of a probabilistic weighting and re-sampling, the algorithm culls all but the top proportion of the new particles (assign them a weight of 1 or 0) based on their ranking according to a heuristic function.

Exploring with insufficient marking aids

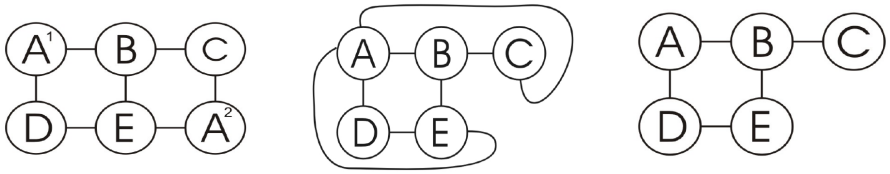
In contrast to the work reviewed above, which assumes no marker, other approaches assume an insufficient supply of markers with which only *some* (but not all) vertices of the graph can be uniquely identified. One such work [69] deals with the problem of *perceptual aliasing* which is caused by repeated structures in the environments. Specifically, the work addresses the problem of inferring a topological map from sequences of deterministic but aliased perceptions. The unknown environment is abstracted as a labelled, but not uniquely labelled, graph in which multiple vertices may share the same label. The vertex labels of the graph represent deterministic (but potentially aliased) sensor readings that characterize places. The paper proposes an approach to infer a topological map from sequences of vertex labels obtained from traversals of the environment.

This approach bears some similarities to the marker-less approach by Dudek et al. [23, 29]: (1) Since some of the vertices cannot be uniquely identified, several possibilities (hypotheses) are examined. (2) Neighborhood structure of a vertex is exploited for disambiguation. If a vertex label is not distinctive, the neighborhood of the vertex is considered in order to disambiguate otherwise identical places. (3) The principle of Occam's razor is used to construct a small map – in terms of vertices – that best explains the observed sequence of labels.

The robot first traverses the environment extensively, painting labels on the vertices and then recording the sequence of painted labels observed during subsequent traversals. Denote the sequence of observed labels as h. Then the neighborhood information, which is not accessible directly from the unknown environment, is obtained from h. The paper defines an n-gram as a length n (contiguous) sub-sequence extracted from the sequence of labels h in which consecutive labels originate from adjacent vertices in the environment graph, and defines *Grams(h,n)* as the set containing all n-grams from the history sequence h. As an example, assume the graph painted by the robot as shown in Figure 2.8(a) and also assume that during subsequent traversal the observed label sequence is h= A-B-C-A-E-D-A-B-E-A-C-B-E-D-A-B-C. Then the set *Grams(h,2)* is {(A,B), (B,C), (C,A), (A,E), (E,D), (D,A), (B,E), (E,A)} and the set *Grams(h,3)* is {(A,B,C), (B,C,A), (C,A,E), ...}. The algorithm infers the map by merging the n-grams using a stochastic local search with respect to the mapping constraints, which includes a *hard* constraint and a *soft* constraint. The *hard* constraint is that the neighborhood of each vertex of the environment graph corresponds to the neighborhood information in the map. That is, the inferred map graph must explain the traversal history. While maintaining the hard constraint, the approach aims to find

a small map, minimizing the number of vertices. This is formulated as a *soft* constraint of the mapping constraints.

The mapping process starts with an empty map graph G_{map} which is augmented during mapping and the set $\Gamma = Grams(h, n)$ which initially contains n-grams extracted from the traversal history h. In the main loop the algorithm selects an n-gram $\gamma \in \Gamma$ and tries to merge it with the map graph G_{map}. A merge is either successful or unsuccessful. A merge is successful if it does not violate the (hard) mapping constraints and is unsuccessful otherwise. In either case, the algorithm proceeds by trying to merge another $\gamma \in \Gamma$ with G_{map} until Γ is empty or, otherwise the possibilities for adding n-grams have been exhausted and the map is aborted. The order in which the γs are selected to be merged is arbitrary. For each selected n-gram γ there might be several possibilities for merging it to G_{map}, resulting in different hypothesis map graphs. To test whether a merge possibility (hypothesis map) is appropriate, the set M of all merge possibilities of merging γ with G_{map} is generated and maintained in ascending order according to the number of vertices in the hypothesis map graph. Then, beginning with a merge possibility (hypothesis map) that requires the fewest vertices, every merging possibility (hypothesis map) $m \in M$ is tested to see whether it satisfies the (hard) mapping constraints. If the hypothesis map satisfies the constraint, the merge is successful and all other merge possibilities (hypothesis maps) which contain more vertices are immediately removed from M due to the soft constraint, otherwise the merge is unsuccessful and m is removed from M and the next merging possibility (hypothesis map) in M is tested; To test whether a merge possibility m satisfies the hard constraint, a set of local n-grams is extracted from the hypothesis map (by virtually 'traversing' the hypothesis



(a) (b) (c)

Figure 2.8: (a) Example environment graph. Also a good hypothesis map. Note the two aliases, labelled A^1 and A^2. (b) An incorrect hypothesis map of (a). (c) Possible partial map G_{map} to be merged. Courtesy of [69].

map), and this is then compared with $Grams(h, n)$. An n-gram that is contained in the set of local n-grams but is not in *Grams(h,n)* indicates a violation of the hard consistency constraint and results in the potential merge being abandoned. For example, consider two possibilities to merge the 3-gram (C,A,E) with the partial G_{map} shown in Figure 2.8(c), which result in the hypothesis maps shown in Figure 2.8(a) and 2.8(b). The potential merging that requires fewer vertices is tested first (Figure 2.8(b)). However, this potential merge violates the mapping constraints and thus is abandoned: the hypothesis map contains some local n-gram (e.g.,(E,A,D) and (C,A,D)) that are not in $Gram(h, 3)$. That is, the traversal history h (on the environment) cannot 'explain' the neighborhood information (E,A,D) and (C,A,D) obtained in the hypothesis map of this merging possibility. The algorithm infers a map graph which is consistent with a given set of n-grams generated from the observation history. The arbitrary selection order of γ from Γ can cause the algorithm to produce large maps or get stuck during mering. Having no way of avoiding this, the algorithm is run repeatedly and returns the smallest valid map found. In the empirical evaluations presented in the paper, the vertices of each graph are labelled with elements from a set whose cardinality corresponds to different fractions (40%–90%) of the cardinality of

the set of vertices. Both 3-grams and 5-grams are evaluated. Whether the inferred map graph is isomorphic to the environment graph is treated as a measure of the quality of the map graphs the algorithm infers. Results show that not all the inferred maps are isomorphic to the environment graph. When the degree of aliasing in an environment does not exceed a certain limit, e.g., for label set cardinality of more than 80% of the vertex set, using both 3-grams and 5-grams the proposed method often finds a topology that is isomorphic to that of the underlying environment. When the aliasing exceeds a certain limit (e.g., for label set cardinality of no more than 60%), then using 5-grams generates about 50%–70% isomorphic maps, whereas using 3-grams typically generates less than 50% isomorphic maps.

Probabilistic approaches to topological mapping

Given the success of probabilistic approaches to metric SLAM, it is not surprising that there has emerged work that directly extends the probabilistic approach used in metric representations to topological mapping by computing the probability distribution over the space of all topological maps. This work includes [16, 14, 58], and a series of papers by Ranganathan et al. [49, 50, 51, 52]. Compared with the approaches above, these probabilistic topological approaches usually incorporate additional measurements such as edge length and even appearance information to define the local node signature.

[58] presents a method for topological SLAM that specifically targets loop closing for edge-ordered graphs. Similar to the marker-less work of Dudek et al. [23, 29], this paper proposes a multi-hypothesis technique that relies on the incremental construction of a map/state hypothesis

tree. Instead of using a ranking heuristic function to evaluate hypotheses as in [29], this paper proposes a probabilistically grounded multi-hypothesis technique in which likely hypotheses are chosen based on their posterior probability after a sequence of sensor measurements. Contributions of the work include the design of a tree expansion algorithm specific to edge-ordered graphs and the introduction of a customized method for recursively computing the posterior probability over the topological map hypotheses. The work also introduces a set of conservative pruning rules that help reduce the size of the hypothesis tree. At time step k, each hypothesis h represents a possible edge-ordered topological graph G_k^h as well as the robot's state X_k^h on that graph. The state is represented by the vertex v_k^h at which the robot is currently located, and the edge e_k^h from which the robot arrived at that vertex, i.e., $X_k^h = (v_k^h, e_k^h)$. The edge-ordered graph G_k^h is represented by the number of vertices N_k^h and a set of circular neighbor lists L_k^h (one list per vertex), thus $G_k^h = (N_k^h, L_k^h)$. A neighbor list $L_k^h(v_k^h)$ stores the vertices in the graph that are neighbors of vertex v_k^h in the order they occur (counter-clockwise from the first mapped edge). An element of the neighbor list $L_k^h(v_k^h, j)$ represents the neighboring vertex of v_k^h along the j-th edge. The goal of the approach is to incrementally build a set of hypotheses that can completely reproduce the possible map/state pairs at every time step k. The approach thus maintains a hypothesis tree where each level of the tree represents a different time step in exploration.

At each time step k, the robot transits to another vertex by choosing a motion input u_k, which is a relative offset from the previous arrival edge. It is assumed that the robot correctly performs the motion input u_k at each time step and therefore leaves the previous vertex via the appropriate departure edge. When the robot chooses a new motion input u_k, the hypothesis

tree is updated by expanding all of the leaf nodes of the tree (the leaf nodes are elements of the set of hypotheses H_{k-1} at time step $k-1$). H_{k-1} leaf nodes of the hypothesis are expanded in the following way: If $L^h_{k-1}(v^h_{k-1}, \beta_k)$ (the neighbor of v^h_{k-1} that is associated to the departing edge β_k) is *explored* then the algorithm copies the hypothesis to a single child hypothesis but moves the robot's state to the new vertex and updates the arrival edge. If $L^h_{k-1}(v^h_{k-1}, \beta_k)$ is *unexplored* then the algorithm considers several possibilities that agree with hypothesis h. The first possibility is that the robot traverses the unexplored edge and arrives at a new vertex (one hypothesis is spawned for this possibility). Additionally, the algorithm considers that a loop is closed and the robot arrives at a previously visited vertex via one of its unexplored edges. One hypothesis is spawned for each unexplored edge in the graph (except for the current departure edge).

To determine which hypotheses among the leaf nodes of the hypothesis tree are likely to represent the true state and the true map, the algorithm computes the posterior probability of each hypothesis given a sequence of sensor measurements. The hypothesis that fits the sensor data better produces a higher probability measure and is therefore more likely to represent the true state and map. During the transition at time step k, a measurement z^e_k is obtained during the edge traversal, which includes a travel distance measurement. Also, a measurement z^v_k is obtained when the robot arrives at the new vertex, which includes a range measurement to obstacles. The posterior probability of a hypothesis is represented as $p(X^h_k, G^h_k | z_{0:k}, u_{1:k})$ where, as before, X^h_k and G^h_k represent the robot's state and graph respectively. $z_{0:k} = (z^v_{0:k}, z^e_{1:k})$ is the collection of all measurements during the experiment, which includes the edge measure-

ment sequence $z^e_{1:k}$ and the vertex measurement sequence $z^v_{0:k}$. The sequence $u_{1:k}$ represents the motion inputs through time step k. Using Bayes' law, the posterior is reformulated to $p(X^h_k, G^h_k|z_{0:k}, u_{1:k}) = \eta p(z_{0:k}|X^h_k, G^h_k, u_{1:k})p(X^h_k, G^h_k|u_{1:k})$ where $p(z_{0:k}|X^h_k, G^h_k, u_{1:k})$ is the measurement likelihood function and $p(X^h_k, G^h_k|u_{1:k})$ is a prior on the hypothesis. The prior is reduced to $p(X^h_k, G^h_k|u_{1:k}) = p(X^h_k|G^h_k, u_{1:k})p(G^h_k|u_{1:k}) = p(G^h_k|u_{1:k})$ because the probability of the state given the map and inputs, $p(X^h_k|G^h_k, u_{1:k})$, is equal to one – due to the assumption that a robot can correctly perform the motion input sequence. The scalar value η is used for normalization over all possible hypotheses such that $\sum_{h=0}^{H_k-1} p(X^h_k, G^h_k|z_{0:k}, u_{1:k}) = 1$, where H_k is the number of current leaf nodes in the hypothesis tree. In computing the measurement likelihood function $p(z_{0:k}|X^h_k, G^h_k, u_{1:k})$ the algorithm maintains for each hypothesis the mean of the measurements associated to each edge, as well as the mean of the measurements associated to each vertex. These means act as sufficient statistics for the history of sensor measurements $z_{0:k-1}$. The algorithm also keeps track of the number of measurements associated with each edge and vertex. The measurements are assumed to have additive zero mean white Gaussian noise with known covariances for the edge and vertex respectively. Using these notations, the likelihood update is computed recursively using a customized method (see [58] for details). For the prior $p(G^h_k|u_{1:k})$, which represents the probability that the robot happens to be placed in an environment with a topology G^h_k (without any sensor information), the authors claim that while there is no way to know the right answer, it is possible to do better than using a uniform distribution: to prevent data over-fitting, the authors use the distributions $p(G^h_k|u_{1:k}) \propto \exp(-N^h_k \log k)$. When two hypotheses have a similar likelihood, this prior gives preference to the smaller map. The authors

claim that by computing the posterior using both the prior described here and the likelihood function described above, the approach captures a balance between small concise maps that would make sense for a structured environment and large maps that better fit the data.

In order to keep the tree size bounded while executing the tree expansion algorithm, a series of pruning tests are applied to the leaf hypotheses at each time step. These include a Degree Test to eliminate hypotheses containing a vertex whose degree does not match the sensed degree, and a Posterior Probability Test, which is used to eliminate any hypothesis whose posterior probability $p(X_k^h, G_k^h | z_{0:k}, u_{1:k})$ drops below a threshold τ, as the low probability implies that the hypothesis is a very poor fit to the sensor data. The paper presents experiments on several environments in which there are a number of ambiguities that make mapping difficult (e.g., vertices that share a similar appearance and edges that are the same length). Despite the ambiguities, the robot correctly maps the environment and at the end of the experiments only one hypothesis survives the pruning steps, and it is the correct state and map. The authors also note that, since Dudek et al.'s marker-less algorithm [23] removes hypotheses in the tree only when the graph becomes inconsistent, if the implementations in [23] were run on the same data set then the number of hypotheses is expected to grow beyond what is computationally feasible.

2.3.3 Summary

Topological SLAM is a key problem in robotics. It enables an understanding of the fundamental limits to SLAM and at the same time provides a framework for the development of SLAM algorithms that operate in the real world. Topological SLAM is not solvable deterministically

without an appropriate marking aid, whereas with a sufficiently powerful marking aid, topological SLAM is solvable deterministically. But what information is sufficient in deterministic topological SLAM, and do different marking aids provide different capabilities? These are the problems considered in the following chapters.

Chapter 3

Exploring topological worlds

This chapter focuses on deterministic exploration where a marking aid is used by the robot. Previous work reviewed in Chapter 2 examined the power of *movable* marker(s) in exploring undirected topological worlds. It was shown in [26] that a single undirected movable marker is sufficient to solve the SLAM problem deterministically. [26] solved the loop closing problem using a single undirected marker that is dropped and picked up at vertices. The marker is used to provide both *place validation*, which determines if two locations are distinct, and *back-link validation*, which determines the relative embedding (orientation) of a given vertex. Is there a need for the marker to provide both explicit place and back-link information? This chapter explores the fundamental limits of exploration and mapping. The problem addressed in this chapter is: given an unknown environment modelled as a graph, can the world be mapped deterministically with a 'simpler' marker? Work in this chapter shows that while a marker providing only place (position) or back-link (local orientation) information in a vertex is not sufficient, mapping is solvable if *both* explicit place and back-link information exist in one vertex of the world. Such information enables the robot to determine the identity of each vertex it is visiting and the back-link (entry edge) by which the robot entered the vertex. Such 'directional

lighthouse' information can be established in a number of ways, for example, through the use of a *directional* immovable marker[4]. This chapter develops a basic deterministic SLAM algorithm that uses a directional lighthouse. The basic algorithm is proved correct and is evaluated in simulation and on a real robot. A number of different mechanisms for establishing a directional lighthouse are discussed.

3.1 Background

3.1.1 Formal world and robot model

The formal world model used in this work is taken from [26] with extensions to allow for other marker classes as described later. The world is modelled as an embedded graph. The goal of the robot's exploration is to build an embedded graph representation that is isomorphic [37] to the world it has been assigned to explore. The robot's inputs are its perceptions and it can only interact with the world through its motions in the world and its operations on the markers (if any).

The world. The world is modelled as an embedding of an undirected graph $G = (V, E)$ with a set of vertices $V = \{v_0, ..., v_{n-1}\}$ and a set of edges $E = \{(v_i, v_j)\}$. Denote the number of edges and vertices in G by $m = |E(G)|$ and $n = |V(G)|$ respectively. Vertices in G correspond to locations in the world and edges in G correspond to connections of the locations. G is an *unlabelled* or *anonymous* graph as vertices and edges of G are not necessarily uniquely distinguishable to

[4]Some of the results in this chapter have already appeared in the literature [63, 64] or are currently submitted to robotic journal and conferences.

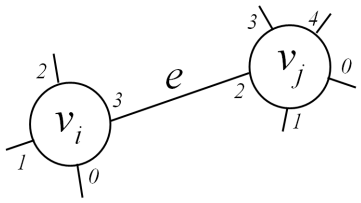


Figure 3.1: An edge and its labels (indices) $l_i = 3$ and $l_j = 2$ at its two end vertices, assuming a planar embedding and a clock-wise enumeration rule.

the robot using its sensors. We restrict the world model to graphs G that contain no cycles of length ≤ 2, i.e., the graph contains no degenerate or redundant paths. This restriction prohibits the world from having multiple edges between two vertices or a single edge incident twice at the same vertex. This also limits the number of edges m to be less than or equal to the square of the number of vertices n.

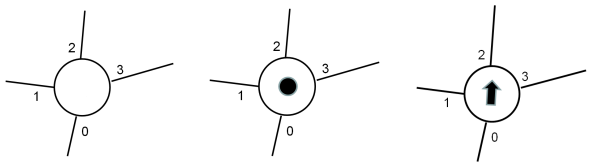
The world G is embedded within some space (not necessarily a planar surface) in order to permit relative directions to be defined on the edges incident upon a vertex. Specifically, the embedding permits the definition of an edge to be extended to allow for the explicit specification of the relative order of edges incident upon each vertex of the graph embedding. This ordering is obtained by enumerating the exits (edges) in a vertex in a systematic manner (e.g., clockwise for planar embedding) from some standard starting direction. An edge $e = (v_i, v_j)$ incident upon v_i and v_j is assigned labels (indices) l_i and l_j, one for each of v_i and v_j respectively. l_i and l_j represent the ordering of the edge e with respect to the consistent enumeration of edges at v_i and v_j respectively. The labels l_i and l_j can be considered as general directions, e.g., from vertex v_i edge (exit) l_i takes edge e to vertex v_j via edge (exit) l_j. An example is shown in Figure 3.1.

Robot motion and edge-related perception. Assume that the robot can leave a vertex by a given exit (edge) of the vertex, can move between vertices by traversing edges, and can identify when it arrives at a vertex. Given the graph embedding described above, a single move of the robot can be specified by the relative order l' of the edge along which the robot exits the current vertex, where l' is defined with respect to the edge along which the robot entered the current vertex or with respect to the (initial) orientation of the robot at the current vertex which is assumed to align with one of the edges of the current vertex. $l' = 0$ identifies the edge through which the robot entered the current vertex. Based on this, a *motion sequence* executed by the robot, denoted by $\mathcal{M}$, can be specified as a sequence of (relative) edge orderings with respect to the entry edges along which the robot enters each vertex. For example, the motion sequence $\mathcal{M}$=(2,3,1), which, assuming a clock-wise edge enumeration rule, denotes 'take the 2nd next edge to the left of the entry edge, then upon arrival take the 3rd next edge to the left of the entry edge, and then upon arrival take the immediate next edge to the left of the entry edge'.

The perception information that the robot acquires consists of *edge-related* perception and *marker-related* perception. With *edge-related* perception, at a vertex the robot enumerates the edges in the vertex by following the pre-defined enumeration rule, thus determining the relative ordering of edges incident on the vertex in a consistent manner. Note that in a planar environment this enumeration might be as simple as enumerating the exits (edges) in a clockwise manner, but more sophisticated enumeration schemes are required for higher dimensional spaces or spaces lacking a gravity-like reference frame. The robot can sense the number of exits (i.e., degree) of the current vertex, can identify the edge through which it entered a vertex and can assign a

relative label (index) to each edge in the vertex, representing the current local edge ordering at the vertex. We assume that the robot follows the same consistent enumeration rule throughout exploration. Entering the same vertex from two different edges will lead to two local edge orderings, one of which is a permutation of the other. For ease of exposition, in this work a planar embedding and a simple clock-wise enumeration rule are assumed in two-dimensional examples. Under this assumption, two local edge orderings at a vertex are cyclic permutations of each other. We do not assume that the robot has mechanism for determining absolute distance and orientation information. Thus vertices are featureless except for the exits to other vertices. Two vertices appear identical to the robot if they have the same degree.

Markers, marker-related operation and perception. The robot is equipped with one or more markers, which are objects that can be used to mark places of interest. The number and type of markers change from one scenario to the next, as described later. A *marker-related* operation enables the robot to manipulate the marker(s) located with the robot or at the current location, and *marker-related* perception enables the robot to sense the presence or absence of the marker(s) at the current location, along with other marker-related information. While detailed operations and perceptions vary for different marker classes as is made clearer in later discussions, the fundamental assumption made in this chapter is that upon entering a vertex, the robot can sense the presence or absence of the marker at the vertex. The marker can either be undirected or directional. For a directional marker, the marker identifies a specific direction by pointing to one of the edges. A directional marker thus provides both presence information and orientation information.



(a) (b) (c)

Figure 3.2: Example signatures. Assume that the robot enters the vertex via edge 0. (a) Signature [4,A,A]. (b) Signature [4,V,A]. (c) Signature [4, V-2, A].

The sensory information that the robot acquires while at a vertex is the pair consisting of both the edge-related perception and marker-related perception. Such sensory information at a vertex defines the signature [40] of the vertex. Each signature consists of the degree information of the vertex as well as marker information at the vertex, which includes the presence or absence of the marker at the vertex, and, depending on the marker class used, other information that the marker provides (e.g., directionality for a directional marker). For ease of exposition, signatures are encoded in the general form of [degree, V#-dir, otherInfo] where 'degree' denotes the degree of the current vertex, followed by the absence (A) or presence (V#-dir) of marker(s) located at the vertex. '#' specifies the number of markers sensed in the vertex, which is omitted in this chapter for brevity since only one marker is used. In the case of a directional marker, 'dir' specifies the direction of the marker(s), represented by the relative ordering of the edge pointed by the marker (relative to the entry edge at the vertex). The vertex marker information is followed by any other marker information that may be present. In this chapter this is represented by an 'A' as no other information is present. Some examples are shown in Figure 3.2. Given the definition of signature, we define the sequence of perception information $\mathcal{P}$ that the robot observes during execution of a motion sequence $\mathcal{M}$. The perception sequence $\mathcal{P}$ is specified by a sequence of signatures of

each vertex visited during execution of a motion sequence[5]. For example, $\mathcal{P}$ = ([2,A,A], [3,A,A], [4,V-2,A]).

Memory. The robot remembers all raw sensory information that it has acquired and all of its motions and operations. Specifically, if the robot has performed steps $0, 1, ..i$, the raw memory of the robot contains the sequence of information obtained at each step. For the i-th step, the robot remembers the signature of the vertex visited at step i including the order of edges incident on the vertex, as well as the motion and operation taken at step i. By "remembering" the motion sequence, the robot may retrace any previously performed motion. We assume that the amount of local memory available with a robot is sufficient to store such information.

3.1.2 Metrics and lower bound

In robotic exploration the cost of physically moving a robot is likely to be several orders of magnitude more expensive (in terms of time, power expended, etc.) than the cost associated with the computational effort. Thus, as in [26] and related work, here we consider physical steps moved in the environment (i.e., the number of edge traversals by the robot) as the cost of the exploration algorithm. Given these metrics, a trivial lower bound $\Omega(m)$ exists for the cost of exploring an unknown graph-like world – the robot must traverse every edge in the environment in the process of exploring; otherwise the robot would not know where all the edges go.

[5]For simplicity, this definition excludes the signature of the initial vertex of the motion sequence.

3.1.3 Sketch of the basic topological mapping approach

The deterministic algorithms developed in this and the following chapters follow the basic approach given in [26]. The algorithms maintain a map representation which is built incrementally as the robot explores, and terminate when there are no unexplored places in the world, producing a map representation of the world model. Specifically, the algorithms maintain an embedded graph-like map representation S of the explored subgraph of the real world model G. S consists of vertices and edges that represent (correspond to) vertices and edges of the explored subgraph of the world model G, respectively. For the purpose of exposition, we denote the mapping from the map S to the real world model G by ϕ. For a vertex v in S, $\phi(v)$ is the real world vertex to which v corresponds, and for an edge e in S, $\phi(e)$ is the real world edge to which e corresponds. The algorithms also maintain the set U of unexplored edges which correspond to edges in the world that have been encountered (in the explored ends) but have not been explored yet. Each edge e in U is incident on a previously explored vertex in S, and the other vertex is as yet unknown. The unknown end may be a new vertex that is not in S, or it may be in S but the robot does not know yet which vertex it is. This 'partial' edge e is held in U to indicate that $\phi(e)$ in G has not been explored yet.

Initially $S = \{v_0\}$ where v_0 corresponds to the initial location of the robot. Incident edges at v_0 are the initial elements of U, corresponding to the incident edges at $\phi(v_0)$. One step of an algorithm consists of selecting (and removing) an unexplored edge $e = (v_k, v_u)$ from U, which represents an unexplored edge $\phi(e)$ in the world, and having the robot traverse the explored subgraph to the known end $\phi(v_k)$ and then following $\phi(e)$ (based on the known index at v_k)

to the unknown end v_u[6]. Upon arrival at the unknown end v_u, the robot needs to solve the loop closing problem. Specifically, the robot must perform 'place validation' to determine if v_u is truly distinct from all the previously visited places or it corresponds to some known vertex in S (i.e., 'where am I entering')? If v_u corresponds to a known vertex $v_{k'}$ in S, then the robot needs to conduct 'back-link validation' to determine which incident edge at $v_{k'}$ corresponds to e (i.e., 'by which exit did I enter)? Using different classes of marking aids, the algorithms conduct validations in different ways. For example, in [26] a single undirected movable marker is used for both validations. If the validations show that the unknown place v_u is distinct from all the previously explored vertices in S, then no new loop is formed (call e an *non-loop edge*). Both v_u and e are added to S, augmenting S by one edge and one vertex ('non-loop augmentation'). All other edges incident on v_u, which correspond to unexplored edges in G and thus require further exploration, are added to U. Note that by following the enumeration rule, the algorithm is free to set the labels (indices) of e and the other incident edges at this new vertex. If, on the other hand, the validation shows that v_u corresponds to the known vertex $v_{k'}$ (place validation) and e corresponds to the unexplored incident edge e' at $v_{k'}$ (back-link validation), then a new loop is formed, which leads the robot from v_k to $v_{k'}$ via e/e' (call e a *loop edge*). Both e and e' represent the edge in G that the robot just explored. In this case S is augmented by the edge $e/e' = (v_k, v_{k'})$ ('loop augmentation'). The index of e at v_k and e' at $v_{k'}$ are used as the indices

[6]To facilitate exposition in this chapter, except when absolutely necessary (e.g., in a correctness proof), we will ignore the mapping between S and the world model. Thus S may also denotes the explored subgraph of G, and v in S also denotes $\phi(v)$. Thus a description such as 'select an unexplored edge e $e = (v_k, v_u)$ from U, and having the robot traverse the explored subgraph to the known end $\phi(v_k)$ and then follow $\phi(e)$ (based on the known index) to the unknown end $\phi(v_u)$' is simply described as 'select an edge $e = (v_k, v_u)$ from U, traverse S to known end v_k, then follows e to unknown end v_u.'

of the new edge at v_k and $v_{k'}$ respectively. Edge e' is also removed from U, since, the edge in G that e' (and e) represents is no longer an unexplored edge. Upon completion of the validation and augmentation steps, the algorithm proceeds to the next iteration in which the above steps are repeated with a newly selected edge from U.

Exploration terminates when the unexplored edge set U is empty. When U is empty, the map S is isomorphic to the world G, provided that validation processes solve the loop closing problem correctly. Following [26], here an extended definition of graph isomorphism is used. Map S and real world model G are said to be isomorphic if and only if they are isomorphic under the usual definition of graph isomorphism ([37]), and in addition for each vertex v of S and each edge e leaving v, $index(e, v) = relabelling(index(\phi(e), \phi(v)))$ where $relabelling$ represents that the edges leaving v and $\phi(v)$ have the same labelling (follow the same pre-defined enumeration rule) but may be labelled starting from different reference edges.

Given this sketch, observe that there are m iterations of the algorithm execution, as there are m edges to be explored and each iteration adds exactly one edge to S. Among the m iterations, some iterations may also add one vertex along with an edge, so there are $n - 1$ executions of non-loop edge exploration and non-loop augmentation. In other iterations only edges are added to S. So there are $m - (n - 1)$ executions of loop edge exploration and loop augmentation. Note that the non-loop edges and their end vertices form a spanning tree of the underlying graph G.

3.2 Is deterministic SLAM possible with explicit place or back-link information only?

Given an unknown graph-like world, what is the fundamental information required for the robot to map it deterministically? We consider different amounts of place and back-link information. We define *explicit place information* as unambiguous information as to the current position of the robot, i.e., the identitiy of the (known) vertex that the robot is entering. We define *explicit back-link information* as unambiguous information concering the back-link through which the robot entered the current place, i.e., the entry edge by which the robot enters the current vertex.

3.2.1 Zero-marker case: is a marker necessary?

An interesting and fundamental question about topological exploration with a mobile robot is: can a robot explore and map an arbitrary graph-like world without any marker? Given the world and robot model discussed earlier, the robot may lack any explicit place and back-link information during exploration and it is straightforward to show that a robot lacking such place and back-link information *cannot* map an arbitrary graph deterministically (see [26]). To see this, consider the graphs shown in Figure 3.3. Given that at each location (vertex) the robot can only sense the degree information of the vertex, whenever the robot enters a vertex, it cannot tell whether the vertex is a new vertex or is one of the vertices it had visited previously, since all the vertices (both unknown and known) have the same signature [2,A,A]. All of the vertices thus appear identical to the robot, even if the degree information of arbitrarily large neighborhoods are taken into consideration. Thus if the robot were to explore the three different unknown

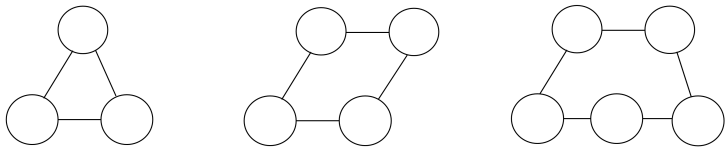


Figure 3.3: Simple indistinguishable graphs. Each vertex appears identical to every other vertex as all have the same signature [2,A,A]. Note that there are infinite number of other graphs that are indistinguishable from these examples.

environments shown in Figure 3.3 it would not be able to tell them apart, even though here we assume perfect robot motion and sensing in identifying a vertex and enumeration of the incident edges. In exploring the graphs, the robot always observes a non-terminating sequence of '2-door rooms' (perception sequence $\mathcal{P}$=([2,A,A],[2,A,A],[2,A,A],...,[2,A,A]).

3.2.2 Exploring with explicit place information only

Can a robot map an arbitrary graph-like world with explicit place information only? Suppose that the environment contains a uniquely marked vertex where explicit place information exists due to the presence of an unoriented (undirected) marker in it, but no explicit back-link information is available. That is, upon entering the marked place, the robot knows which node it is in but cannot determine the entry edge to the place. Suppose further that this is an *immovable* marker which can be recognized when found, but which remains in that location during the exploration. While a single undirected movable vertex marker is sufficient to map all embedded graphs deterministically [26], whether or not a single *immovable* marker can solve the general exploration problem deterministically is an interesting problem that has been raised in the literature [24, 27].

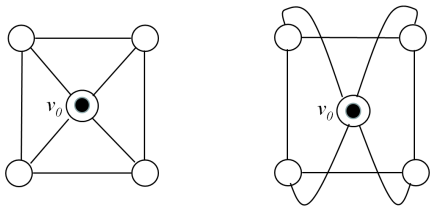


Figure 3.4: Two different embedded graphs that are not always distinguishable with an undirected immovable vertex marker. Assume a clockwise enumeration rule, and that the robot starts from the marked vertex v_0, facing one of the edges. Identical motion sequences always result in same perceptions on both graphs. E.g., motion sequence $\mathcal{M} = (x, 1, 2, 2, 1)$ where $x \in \{0,1,2,3\}$ results in perception $\mathcal{P}$=([3,A,A], [3,A,A], [3,A,A], [3,A,A], [4,V,A]) on both graphs.

In [24] the authors conjectured that if a single undirected immovable vertex marker is used, then the class of graphs that can be successfully explored is reduced as the robot cannot explore all environments successfully.

We justify here that a single undirected immovable marker is *not* always sufficient to solve the mapping problem deterministically[7]. Given that only one location of the underlying world is marked thus explicit place information exists, while the robot can easily distinguish graphs such as different sized cycles, there exist different (embedded) graphs that the robot cannot distinguish deterministically. We justify this by showing that there exist different[8] graphs on which, given the same motion sequence, the perception sequence the robot obtains are the *same*. Consider the two (embedded) graphs shown in Figure 3.4, which are not isomorphic to each other according to the extended definition of graph isomorphism. Each of the graphs is marked with an undirected immovable marker. Assume that initially the robot is located at the marked vertex v_0 and faces

[7]This result appears in [64].

[8]We consider two graphs to be different if they are not isomorphic to each other under the extended graph isomorphism discussed earlier.

one of the edges of v_0. It can be proved that identical motions result in the same perceptions on the two graphs.

For the ease of exposition, suppose that there are two robots each exploring one of the graphs and that both of the robots are initially located at the marked nodes. Call the robot operating on the graph in Figure 3.4(a) robot L, and the robot on the other graph robot R. Within the embedding L and R represent different worlds. We distinguish two types of vertices based on their signatures. Call the marked vertex the *center* vertex and other vertices the *corner* vertices. Now we prove by induction that in executing identical motions, the perceptions obtained by the robots are always the same.

Let $P(n)$ be the statement that after n steps of motion execution (edge traversals), the perception sequences obtained by the robots are the same. Moreover, the neighbor degree lists, enumerated from the current entry edges, are the same.

Base Case: When $n = 1$, both the robots go from the center vertex to one of the corner vertices. The perception obtained by the robots is [3,A,A]. Moreover, starting from the current entry edge and following the clockwise enumeration rule, the enumerated degree list is [4,3,3] for both robots. So $P(1)$ is trivially correct.

Induction hypothesis: Assume that $P(k)$ is correct for some positive integer $k > 1$. That is, after k steps of motion execution, the perceptions obtained by the robots are the same. Moreover, the neighbor degree lists enumerated from the current entry edges are the same.

Induction step: We now show that $P(k+1)$ is correct. That is, after $k+1$ steps of motion execution, the perception sequences obtained by the robots on the two graphs are the same,

and moreover, the neighbor degree lists enumerated from the current entry edges are the same. Given the hypothesis that $P(k)$ is true, it is sufficient to show that the perceptions (signatures) obtained in step $k+1$ are the same on the two graphs, and moreover, at the new places the enumerated neighbor degree lists are the same.

Consider different situations by the end of step k (i.e., before making the $k+1$ traversal). Since the perceptions obtained on the two graphs in the first k steps are the same, the robots are either both at the center nodes (Case 1), or both at corner nodes (Case 2).

Case 1. At the end of step k, both robots are in the center nodes on their graphs. In step $k+1$ both the robots move to corner nodes of their graphs, obtaining the same perception [3,A,A] and the same enumerated degree list [4,3,3].

Case 2. At the end of step k, both robots are in corner nodes. We show that identical motions from the current places lead the robots to the same type of vertices (center nodes or corner nodes), where the enumerated degree lists are identical. We distinguish three sub-cases based on the robots' entry edges in step k.

(2.1) In step k, robot L entered a corner node via the edge connecting to center node. That is, robot L was in the center node after step $k-1$. Without loss of generality, this situation is shown in the left half of Figure 3.5(a), where the robot moves to node a in step k via edge (x, a). By induction $P(k-1)$ is true, robot R was also in the center node and thus in step k robot R also enters a corner node via the edge connecting to the center node. Without loss of generality, this situation is

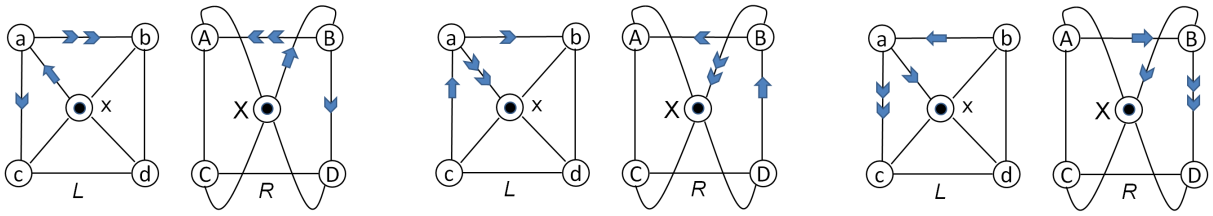


Figure 3.5: Entering corners at step k. Label vertices for L as $a-d$ and vertices for R as $A-D$. In (a) entering corners from edges connecting the center nodes. In (b) entering corners from edges that are the 1st edge next to the edges connecting to the center nodes. In (c) entering corners via edges that are the 2nd next to the edges connecting to the center nodes.

shown in the right half of Figure 3.5(a), where the robot enters node B in step k via edge (X, B). We can see now that in step $k+1$, motion 1 drives robot L to corner node c via edge (a, c) and drives robot R to corner node D via edge (B, D). For both of them the perception [3,A,A] and the enumerated degree list [3,4,3] are obtained. Motion 2 drives robot L to b via edge (a, b) and drives robot R to A via edge (B, A), where the same perception [3,A,A] and the same enumerated degree list [3,3,4] are obtained.

(2.2) In step k, robot L entered a corner node via the edge that is the 1st next to the edge connecting to the center nodes. Without loss of generality, this situation is shown in the left half of Figure 3.5(b), where the robot enters node a in step k via edge (c, a). By $P(k)$ is true, robot R must also enter a corner node via the edge that is the 1st next to the edge connecting to the center node (so that the enumerated degree list at step k is [3,3,4] for both the robots). Without loss of generality, this situation is shown in the right half of Figure 3.5(b), where the

robot enters node B in step k via edge (D, B). Now for both of the robots, step $k+1$ motion 1 drives robot L to node b via edge (a, b) and drives robot R to node A via edge (B, A). Both the robots obtain the same perception [3,A,A] and the same enumerated degree list [3,3,4]. Motion 2 drives robot L to the center node x via edge (a, x) and drives robot R to the center node X via edge (B, X), where the same perception [4,A,A] and the same enumerated degree list [4,4,4,4] are obtained.

(2.3) In step k, robot L entered a corner node via the edge that is the 2nd next to the edge connecting to the center nodes. Without loss of generality, this situation is shown in the left half of Figure 3.5(c), where the robot enters node a via edge (b, a). By induction $P(k)$ is true, in step k robot R must also move to a corner node via the edge that is the 2nd next to the edge connecting to the center node (so that the enumerated degree list is [3,4,3] for both the robots). This situation is shown in the right half of Figure 3.5(c), where the robot enters node B via edge (A, B). Now for both of the robots, step $k+1$, motion 1 drives them to their center nodes, where the same perception [4,A,A] and the same enumerated degree list [4,4,4,4] are obtained. Motion 2 drives robot L to node c via edge (a, c) and drives robot R to node D via (B, D). For both of them the same perception [3,A,A] and the same enumerated degree list [3,4,3] are obtained.

So in step $k+1$, the robots visit the same type of vertices, obtaining the same signatures and the same enumerated neighbor degree list. So after $k+1$ steps of motion execution, the obtained

perceptions are the same, and the enumerated degrees are the same. Hence, $P(k+1)$ is true.

Therefore, deterministic SLAM is not always possible with a single undirected immovable vertex marker, which provides explicit place information in a vertex but no explicit back-link information.

3.2.3 Exploring with explicit back-link information only

Can a robot map an arbitrary graph-like world deterministically with some explicit back-link information but no mechanism providing explicit place information? Suppose that on a cycle graph the robot has explicit back-link information but no place information so it can determine its entry edge at each vertex it is visiting but not its location. Clearly the robot still cannot distinguish cycles of different lengths. So a robot with explicit back-link information only cannot always map an arbitrary world deterministically. Therefore topological SLAM is not always possible with explicit back-link information alone.

3.3 Exploring with both explicit place information and back-link information

Given that a single undirected movable marker is sufficient to map a topological environment deterministically, but that neither explicit back-link nor a single location providing explicit place information (e.g., in a location marked with an undirected immovable marker) is sufficient, can a world be mapped deterministically with a 'simpler' marker? This section shows that if the world contains a vertex that provides a unique signature that also provides explicit back-link information, then the world can always be mapped deterministically. More specifically, it is

demonstrated that a unique *directional lighthouse* vertex is sufficient to enable deterministic SLAM. A directional lighthouse vertex is defined as a vertex that provides both explicit place information (this is a unique location in the environment) and which at the same time provides explicit back-link information at that vertex. Upon entering a directional lighthouse vertex, the robot knows the identity of the vertex it is visiting as well as the entry edge by which it entered the vertex. The directional lighthouse can be established by marking a location in the world with an immovable marker that provides explicit back-link information, or in a number of other ways as will be discussed later. For the purpose of exposition assume that the robot has a *directional* immovable marker that can be dropped in one vertex of the world. Assume that the robot drops the marker at its initial vertex v_0 and points the marker head toward one of the exits (edges). Then whenever the robot returns to v_0, by enumerating the edges and identifying the one that is pointed to by the marker head, the robot is able to distinguish different edges at v_0, based on the relative ordering between the edges and the marked edge. That is, a directional vertex marker not only identifies the unique vertex in which it is dropped (explicit place information) but it also provides the absolute edge ordering (explicit back-link information) at that vertex. This section shows that with *both* explicit place and back-link information in a vertex, the robot can map an arbitrary topological world deterministically[9].

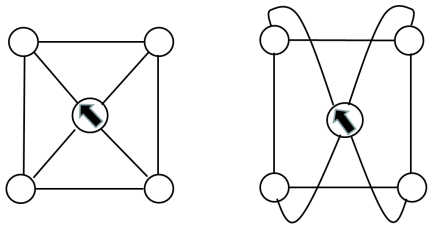
An interesting question here is in what sense is a marker the minimum marker, and is a single immovable marker the minimum marker? It is difficult to define minimum of markers in the global sense as there are many different dimensions to characterize the sophistication of markers.

[9]This result appears in [63] and [64].

Given that a single movable marker is sufficient to solve the SLAM problem deterministically, here we consider the number of markers as the main dimension of marker complexity. Then for a given number of markers, we consider the movability of the marker, which indicates the amount of robot operations on it. Given this local definition, a single immovable marker, which does not involve robot operation on it during exploration, is considered simpler than a single movable marker, and is regarded as the minimum marker here. It is shown above that a single immovable marker is not always sufficient to solve the SLAM problem deterministically if it contains no direction information. As shown in this section, a single immovable marker augmented with direction information becomes sufficient.

In order to provide some intuition as to how a directional lighthouse can provide more information for disambiguating locations, here we revisit the example given in Figure 3.4 but with a directional immovable marker (Figure 3.9). Assume again that initially the robot is located at vertex v_0, and its orientation is in the direction of the marker head. Identical motion sequences now lead to different perception sequences on the different graphs. For example, motion sequence $\mathcal{M}$=(0,1,1) leads to perception sequence $\mathcal{P}$=([3,A,A], [3,A,A], [4,V-1,A]) on the left graph and $\mathcal{P}$=([3,A,A], [3,A,A], [4,V-3,A]) on the right graph. Another example is given in Figure 3.9.

Formally, here we present a deterministic algorithm that maps the world with a directional lighthouse established using a single directional immovable marker in a vertex. This algorithm only needs to be modified slightly if the directional lighthouse is established in other ways. Following the general algorithm sketch given above, this algorithm maintains a map representation S of the explored subgraph of the world, and an unexplored edge set U. Initially $S = \{v_0\}$ where



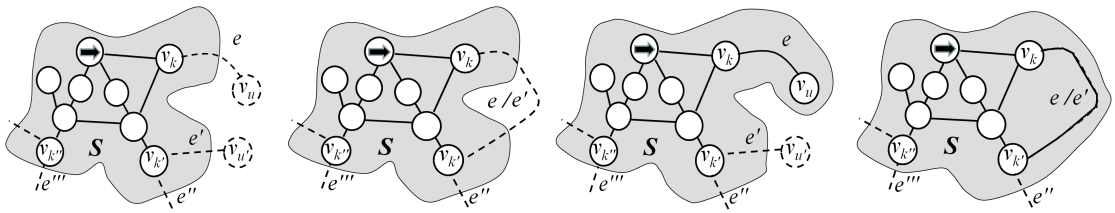
(a) (b)

Figure 3.6: Two different embedded graphs that are not distinguishable with an undirected immovable vertex marker but are distinguishable with a directional immovable vertex marker. Assume a clockwise enumeration rule, and that the robot starts from the marked vertex with initial orientation facing the edge pointed to by the marker head. Motion sequence $\mathcal{M}$=(0,1,2,2,1) where 0 means the robot starts by traversing the edge it is facing results in different perception sequences $\mathcal{P}$=([3,A,A], [3,A,A], [3,A,A], [3,A,A], [4,V-1,A]) and $\mathcal{P}$=([3,A,A], [3,A,A], [3,A,A], [3,A,A], [4,V-3,A]) on the graphs.

v_0 corresponds to the initial location of the robot. Incident edges at v_0 are the initial elements of U. The robot enumerates edges incident at v_0 and sets labels of the corresponding edges on S based on the enumerated edge ordering. Choosing v_0 as the directional lighthouse vertex during exploration, the robot drops the directional marker at v_0, pointing the marker toward one of the exits (edges) of v_0 and remembering the label of the exit.

After initialization, each step of the algorithm consists of selecting (and removing) an unexplored edge $e = (v_k, v_u)$ from U, having the robot traverse S to the known vertex v_k and then following e to the unknown end vertex v_u, as shown in Figure 3.7(a). At v_u the robot senses the signature of v_u, which includes the degree information and the presence or absence of the marker as well as the direction indicated by the marker if it is present. If the marker is sensed at v_u, then we know that the robot is coming (back) to v_0 where both explicit place and back-link information exist. Then based on the relative ordering between the entry edge and the edge pointed to

by the marker, the label (index) of the entry edge at v_0 can be identified. Thus both the place and back-link validations are solved without any further motion by the robot, and the algorithm conducts loop-augmentation immediately. A new edge (v_k, v_0) is added into S and the entry edge is removed from U. The index of e at v_k and the index of the entry edge at v_0 are used as the indices of the new edge at v_k and v_0, respectively. If no marker is sensed at v_u, then v_u and v_0 are distinct and the robot may need to do place validation and back-link validation with motion. Every vertex (except v_k) in S could potentially correspond to v_u if (1) it has the same signature as v_u and (2) it has unexplored edge(s). Each unexplored edge incident on such a vertex could potentially correspond to e. Based on this, the key observation is that robot position and back-link (entry edge) can be hypothesized at the same time, and then place validation and back-link validations are conducted *simultaneously* by disambiguating the edge e and the unknown end v_u against unexplored edges currently in U and their known ends. Specifically, each unexplored edge $e' = (v_{k'}, v_{u'})$ in U along with its known end $v_{k'}$ is considered a potential loop closing hypothesis if $v_{k'}$ has the same signature as v_u. Denote the loop closing hypothesis as $h' = (e', v_{k'})$. That is, it is hypothesized that $e = (v_k, v_u)$ and $e' = (v_{k'}, v_{u'})$ correspond to the same edge in the world – v_u corresponds to $v_{k'}$ and $v_{u'}$ corresponds to v_k – and thus the robot has entered $v_{k'}$ from v_k via e' (Figure 3.7(b)). If no such loop closing hypothesis exists, then no loop is formed and the algorithm moves on to the augmentation stage of e as shown later. Otherwise, the hypothesis validation process for e and v_u starts. For each hypothesis $h' = (e', v_{k'})$, a motion sequence $\mathcal{M}_{h'}$ for a simple path on S from $v_{k'}$ to v_0 is computed. Should h' be true, $\mathcal{M}_{h'}$ would drive the robot through S from the current place v_0 without encountering repeated vertices. Hypothesizing that



(a) Traverse to v_u via e (b) A loop closing hypothesis (c) All hypotheses fail. (d) Hypothesis is confirmed

Figure 3.7: A single directional immovable vertex marker algorithm. S is augmented in (c) and (d). Dotted lines represent the unexplored or hypothesized portions of the graph-like world, and solid lines represent the explored portion of the world.

the robot is now in $v_{k'}$ via entry edge e', motion sequence $\mathcal{M}_{h'}$ consists of a sequence of relative edge orderings at each vertex visited along the motion execution, including the ordering at initial vertex $v_{k'}$ (relative to the known ordering of e' at $v_{k'}$). Assuming a clockwise enumeration rule, an example motion sequence for hypothesis $h' = (e', v_{k'})$ in Figure 3.7(b) is (2,2,1) which means 'traverse the 2nd next edge to the left (of current entry edge e') to the other end, upon arrival take the 2nd next edge to the left of the entry edge, and then upon arrival take the 1st edge to the left of the entry edge, move to the other end (v_0)'. The expected perception sequence that the robot should obtain along the execution of $\mathcal{M}_{h'}$, denoted $\mathcal{P}^E_{h'}$, is also computed. $\mathcal{P}^E_{h'}$ is a sequence of signatures of the vertices that should be encountered during execution of $\mathcal{M}_{h'}$ if hypothesis h' is true. The expected perception sequence of the motion sequence $\mathcal{M}_{h'} = (2,2,1)$ should be ([4,A,A], [2,A,A], [3,V-2,A]) where [3,V-2,A] means that (at the end of the path) the marker is present and the marker-pointed edge is the 2nd edge to the left of the entry edge. The key to the correctness of the validation process is the fact that motion sequence $\mathcal{M}_{h'}$ along with the expected perception $\mathcal{P}^E_{h'}$ together define an embedded path $v_0, ..., v_{k'}, v_{u'}$ in S, which

uniquely identifies the hypothesized place $v_{k'}$ and back-link (entry edge) e' specified by h'. Other hypotheses cannot have the same embedded path as h': they may have the same motion sequence as $\mathcal{M}_{h'}$ or may have the same expected perception as $\mathcal{P}^E_{h'}$, but not both. Formal justification for this is given in Section 3.4.

Hypothesized to have entered $v_{k'}$ via e', the robot then validates hypothesis h' by attempting to execute the motion sequence $\mathcal{M}_{h'}$, which, if h' holds, would allow the robot to complete the traversal and obtain perception seqence $\mathcal{P}_{h'}$ that matches $\mathcal{P}^E_{h'}$. That is, the robot would start from $v_{k'}$ and arrive at v_0 via the expected entry edge. The robot obtains real perception $\mathcal{P}_{h'}$ during the execution of $\mathcal{M}_{h'}$, compares $\mathcal{P}_{h'}$ against $\mathcal{P}^E_{h'}$, and distinguishes three cases:

(1) A mismatch between $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ is observed during execution of $\mathcal{M}_{h'}$. Specifically, the marker is encountered at some point along the execution of the edge traversal prior to completion, or, the traversal cannot be followed completely due to a mismatch between the sensed degree of the physical vertex and the expected degree of the vertex on the planned traversal.

(2) A mismatch between $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ is observed at the end of execution of $\mathcal{M}_{h'}$. That is, the path is completed but upon completion of the traversal, the marker is not present, or it is present but the marker-pointed edge does not have the expected ordering relative to the entry edge.

(3) $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ match both during and at the end of execution of $\mathcal{M}_{h'}$. That is, upon completion of traversal, the marker is present and the marker-pointed edge has the expected ordering relative to the entry edge.

Once a mismatch between $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ is detected, the robot stops traversing the path immediately. In case (1) the hypothesis h' is rejected, due to the fact that if h' holds then $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ should match throughout the execution of $\mathcal{M}_{h'}$ and thus the robot should be able to follow the planned traversal. Also as the planned trip contains no repeated vertices, the robot should not encounter the marker prior to v_0. The hypothesis is also rejected in case (2). In this case the robot did not arrive at v_0 or did arrive at v_0 but from an unexpected entry edge. Once a hypothesis is rejected, the robot retraces its steps by executing the reverse of the motion sequence, which moves the robot back to v_u and resumes the original 'orientation' at v_u (aligned with the original entry edge e), and then tests one of the remaining hypotheses for e (if any). In case (3) the hypothesis is confirmed and all remaining hypotheses are disregarded (if any). The validation process for e thus terminates either when a hypothesis is confirmed, or, all hypotheses have been tested and rejected. Then the algorithm moves on to the augmentation stage.

If no hypothesis exists or if all of the hypotheses of e are rejected, then v_u does not correspond to any previously visited vertex. Both e and v_u, which represent the newly explored edge and its end vertex, are added to S (non-loop augmentation), augmenting S by one edge and one vertex (Figure 3.7(c)). Other edges incident on v_u are added to U. The algorithm is free to set the labels of edges incident in v_u and the algorithm labels e as the 0'th edge and the others incident edges are labelled according to the enumeration rule. If a hypothesis $h' = (e', v_{k'})$ is confirmed (case 3), then v_u corresponds to the known vertex $v_{k'}$ (place validation) and e corresponds to the incident edge e' at $v_{k'}$ (back-link validation). In this case the algorithm augments S with a new edge $e/e' = (v_k, v_{k'})$, using the index of e at v_k and index of e' at $v_{k'}$ as the index of the new

edge at v_k and $v_{k'}$ respectively (Figure 3.7(d)). e' no longer represents an unexplored edge in the world so it is removed from U. The algorithm then proceeds with a newly selected unexplored edge from U, and terminates when U is empty. We justify below that when U is empty, the map S is isomorphic to the world G. The algorithm is sketched in Algorithm 3.1.

3.4 Correctness sketch of the single directional immovable marker algorithm

Lemma 1: A hypothesis $h' = (e', v_{k'})$ is uniquely identified by its motion sequence $\mathcal{M}_{h'}$ and the expected perceptions $\mathcal{P}^E_{h'}$.

Proof. We show that for a hypothesis h', the motion sequence $\mathcal{M}_{h'}$ along with the expected perceptions $\mathcal{P}^E_{h'}$ is unique to h'. Other hypotheses may have the same motion sequence as h' or may have the same expected perception as h', but not both. First see that the motion sequence $\mathcal{M}_{h'}$, which encodes (at the beginning) the ordering of e' at $v_{k'}$, along with the expected perceptions $\mathcal{P}^E_{h'}$ which encodes (at the end) the ordering of the entry edge at v_0, specifies an embedded path $v_0, ..., v_{k'}, v_{u'}$ in S. Clearly this embedded path, which should be traversed successfully if h' is true, (uniquely) identifies the hypothesized place $v_{k'}$ and entry edge (back-link) e'. Now we show that this path is uniquely encoded by $\mathcal{M}_{h'}$ and $\mathcal{P}^E_{h'}$. It is trivially true that other hypotheses must have paths different from that of h'. Now we show that each of the other paths must be encoded by a different motion sequence or a different expected perception, or both. If the path for a hypothesis h'' enters v_0 from a different entry edge than the entry edge of h', then the expected perception $\mathcal{P}^E_{h''}$ must be different from $\mathcal{P}^E_{h'}$, although h' may have the same motion sequence as $\mathcal{M}_{h'}$. If, on the other hand, the path for a hypothesis h'' enters v_0

Algorithm 3.1: Mapping with a directional immovable vertex marker

Input: the starting location v_0 in G; a directional marker
Output: a map representation S that is isomorphic to world G

the robot drops the directional marker at v_0, pointing toward one of the edges;
$S \leftarrow \{v_0\}$; *// initial S*;
$U \leftarrow$ incident edges in v_0; *// initial U*;
while *U is not empty* **do**
    remove an unexplored edge $e = (v_k, v_u)$ from U;
    the robot traverses S to v_k and then follows e to v_u;
    the robot senses the signature (degree, marker presence and direction) at v_u;
    $H \leftarrow$ set of loop closing hypotheses of unexplored edges (in U) and their known end vertices which have the same signature as v_u;
    while *H is not empty* **do**
        $h' = (e', v_{k'}) \leftarrow$ a hypothesis removed from H;
        compute a simple motion sequence $\mathcal{M}_{h'}$ which drives the robot from $v_{k'}$ to v_0;
        compute the expected perception $\mathcal{P}^E_{h'}$ of $\mathcal{M}_{h'}$;
        the robot attempts to execute $\mathcal{M}_{h'}$;
        based on the perception information $\mathcal{P}_{h'}$ obtained in executing $\mathcal{M}_{h'}$ **do**
            case *(1) or (2) – $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ do not match throughout*
                reject the hypothesis;
                the robot retraces its motion sequence, coming back to v_u, aligning to the original entry edge e;
            case *(3) – $\mathcal{P}_{h'}$ and $\mathcal{P}^E_{h'}$ match throughout execution of $\mathcal{M}_{h'}$, path completed*
                confirm the hypothesis and exit the inner 'while' loop;
    // now do augmentations on S;
    if *a hypothesis $h' = (e', v_{k'})$ is confirmed* **then**
        // do loop augmentation;
        add edge $e/e' = (v_k, v_{k'})$ to S; *// uses existing indices at v_k and $v_{k'}$*;
        remove e' from U;
    else *// no hypothesis exists, or all the hypotheses are rejected;*
        add e and v_u to S; *// non-loop augmentation*;
        add other edges in v_u to U;
return S;

from the same entry edge as the entry edge of h', then the expected perception $\mathcal{P}^E_{h''}$ may be the same as $\mathcal{P}^E_{h'}$ (e.g., for $h'' = (e'', v_{k'})$ where e'' is another edge on $v_{k'}$), but its motion sequence $\mathcal{M}_{h''}$ must be different from $\mathcal{M}_{h'}$, either from the beginning or in the middle. Hence, the motion sequence $\mathcal{M}_{h'}$ for a hypothesis h' along with the expected perceptions $\mathcal{P}^E_{h'}$ uniquely encode an embedded path $v_0, ..., v_{k'}, v_{u'}$ in S, which in turn uniquely identify the hypothesis h'. □

Following the proof sketch given in [24], now we prove that when the single directional vertex marker algorithm terminates, the map S it maintains is isomorphic to the world model G. We use the extended definition of graph isomorphism described earlier. We prove the algorithm correct by establishing an invariant **I** and showing that **I** is initially true, is maintained true throughout execution, and that the algorithm terminates. Then we show that the termination condition plus the invariant imply the correctness of the algorithm. That is, S is isomorphic to the world model G. We define **I** as follows:

I-1 S is isomorphic to G_s, which is the explored subgraph of the real world model G

I-2 The unexplored edge set U contains a set of edges that may be bijectively mapped to the set of unexplored edges in G that have at least one incident vertex in G_s, with edge indices with respect to the end vertices in S satisfying the edge-index condition.

We also define a bound function $t = |E_G| - |E_S|$ for the loop, where E_G and E_S are the set of edges in G and S respectively. $|E|$ denotes the cardinality of set E.

Proof. **1. I is true before the loop is entered.** Before the loop starts, the explored subgraph G_s of G consists of the single (starting) vertex $\phi(v_0)$, and S consists of the single vertex v_0. Thus,

S and G_s are isomorphic, maintaining **I-1**. U is initialized with the edges that correspond to the edges leaving $\phi(v_0)$. Both set of edges are indexed using the same ordering convention and so the edge-index condition holds. This maintains **I-2**.

2. I is maintained by the loop body. Each time through the loop body, an edge e is selected and removed from U, through which the robot traverses to the unknown end v_u. Then every (other) unexplored edge $e' = (v_{k'}, v_{u'})$ and its known end $v_{k'}$ is considered as a potential loop closing hypothesis if $v_{k'}$ has the same signature as v_u. For each hypothesis $h' = (e', v_{k'})$, a simple motion sequence $\mathcal{M}_{h'}$ is computed, which, if h' is true, could drive the robot from $v_{k'}$ to v_0, The expected perceptions $\mathcal{P}^E_{h'}$, which, if h' is true, would be encountered while executing the motion sequence, is also computed. By Lemma 1, $\mathcal{M}_{h'}$ along with $\mathcal{P}^E_{h'}$ *uniquely* identifies the hypothesized place $v_{k'}$ and entry edge (back-link) e'. This implies that when a hypothesis $h' = (e', v_{k'})$ is accepted (according to case 3), the robot has traversed an edge $\phi(e')$ which leads the robot to another vertex $\phi(v_{k'})$ that is also in G_s. On the other hand, if no hypothesis exists or all the hypotheses are rejected, then no unexplored edges out of G_s corresponds to $\phi(e)$ and thus the robot must have traversed an edge which leads it to a vertex $\phi(v_k)$ that is not in G_s.

In the case that no hypothesis exists or otherwise all the hypotheses are rejected, which indicates that the other end of the vertex $\phi(v_u)$ is not in G_s, both $\phi(e)$ and $\phi(v_u)$ become part of the explored subgraph G_s. Correspondingly, e and v_u are added to S. We need to show that e is labelled correctly, and U is updated correctly. By invariant **I**, $index(e, v_k)$ satisfies the edge-index condition. The algorithm is free to set $index(e, v_u)$ arbitrarily, since no indices of edges leaving v_u has been set yet. The algorithm sets the edge indices in v_u (including that of e) using the

enumeration rule to satisfy the edge-index condition, maintaining **I-1**. In this case, two updates occur to the unexplored edges in G that have at least one incident vertex in G_s: 1) $\phi(e)$ is no longer one of such an edge, as it is explored now. 2) since $\phi(v_u)$ is added to G_s, all untraversed edges incident in $\phi(v_u)$ become such edges. Correspondingly, two updates are made to U: 1) e is removed from U. Note that since v_u was not in S before the loop on this occasion, e was the only entry in U that corresponds to $\phi(e)$. 2) edges incident in v_u are added into U. These edges satisfy the edge-index condition. These updates to U maintain **I-2**.

In the case that a hypothesis $h' = (e', v_{k'})$ is accepted, which indicates that the other end vertex $\phi(v_u)$ of $\phi(e)$ is already present in G_s, $\phi(e/e')$ becomes a new edge but not the 'unknown' end vertex. Correspondingly, edge $e'/e = (v_k, v_{k'})$ is added to S, and e' is removed from U. We need to show that e/e' is correctly labelled with respect to v_k and $v_{k'}$, and U is updated correctly. The algorithm uses the index of e at v_k as the index of the new edge e/e' at v_k, which satisfies the edge-index condition, by the invariant **I-1** is true. The algorithm uses the index of e' at $v_{k'}$ as the index of e/e' at $v_{k'}$, which again, satisfies the edge-index condition, by the invariant **I-1** is true. So the new edge is indexed at its two end vertices correctly. This maintains **I-1**. Since no new vertex is explored, no new untraversed edges are generated for G_s. The only update concerning untraversed edges in G_s is that $\phi(e/e')$ is no longer an untraversed edge in G with at least one incident vertex in G_s. Before the execution of the loop body on this occasion, there must have been two unexplored edges in U (i.e., e and e') that correspond to $\phi(e/e')$. Both the two edges are removed from U by the loop body, so U is updated correctly, maintaining **I-2**.

3 The loop terminates. The loop invariant asserts that S and G_s are isomorphic, so

$|E_S| = |E_H|$. Since G_s is the explored subgraph of G, it only contains edges that are also in G thus $|E_H| \leq |E_G|$, and therefore $|E_S| \leq |E_G|$. This implies that the bound function $t = |E_G| - |E_S|$ must be non-negative. In each iteration one edge is included into S. So in each iteration of the loop body, $|E_S|$ is increased and thus t is decreased ($|E_G|$ is fixed). So the loop must terminate eventually, as t can only remain non-negative for a finite number of iterations through the loop.

4 When the loop terminates, $G_s = G$. We show that when the loop terminates, i.e., when U={}, there are no unexplored edges in G. Assume, to the contrary, that when the loop terminates (i.e., when $U = \{\}$), there exists at least one untraversed edge in G. By invariant **I** and the termination condition $U = \{\}$, the edge must not have unexplored end(s) in G_s. Now assume v is one of these unexplored vertices. Since G is connected, there must be a path from the starting vertex $\phi(v_0)$ to v. This requires that there is an edge on this path with one explored end and one unexplored end. By invariant **I**, there must be a corresponding edge in U, contradicting the termination condition that $U = \{\}$. So there are no unexplored edges in G. This also implies that there are no unexplored vertices in G. That is, the explored subgraph $G_s = G$. So the maintained map S, which is isomorphic to G_s, is now isomorphic to G. □

3.5 Performance of the single directional immovable marker algorithm

3.5.1 Lower cost bound

The lower cost bound for the topological exploration and mapping problem is $\Omega(m)$. This lower bound is tight for the basic single directional immovable marker algorithm. On some environments such as cycles, the algorithm has an exploration cost of exactly m. When running

the algorithm on cycles, each newly explored (unmarked) location is disambiguated against known locations immediately because each of the known locations is either fully explored (i.e., contains no unexplored edges) or contains the marker. Eventually the robot comes back to the starting location, which contains both explicit place and back-link information. Similarly, running the algorithm from one end of a chain has exactly m exploration cost[10].

3.5.2 Upper cost bound

We begin by bounding the number of edge traversals required in one pass through the loop body. Let n_s be the number of vertices in S during the current execution of the loop body. Each loop begins with the robot traversing the graph to v_k and then following edge e to v_u. This traversal requires at most n_s edge traversals. Then for each potential loop closing hypothesis of e, we validate it by having the robot traverse a simple path to v_0. Either all the hypotheses are rejected (v_u is a new place and e is a non-loop edge) or one of the hypotheses is accepted (v_u is one of the known places and e is a loop edge). There are $n-1$ iterations of the loop in which all of the hypotheses are rejected and thus S grows by a non-loop edge and a new vertex, and in each of the remaining $m-n+1$ iterations one of the hypotheses is accepted and S grows by a loop edge. A worst case scenario would see S growing to its full number of vertices in the first $n-1$ iterations. In each of these iterations all the hypotheses are examined and rejected. In the worst case scenario all of the current unexplored edges incident on non-marked places are

[10]When running the algorithm from somewhere in the middle of a chain, less than $2m$ cost is required. The robot traverses to one end and then turns back to the other end. So totally $m+d$ edge traversals are required where d is the 'distance' to the first end it encountered. Since $1 \leq d \leq m-1$, the total cost range from $m+1$ to $2m-1$.

potential hypotheses, and each hypothesis is rejected at the end of path execution. Each path traversal is bounded by $2(n_s - 1)$, and we can bound the number of hypothesis by $2m - 2(n_s - 1)$ where $(n_s - 1)$ represent the number of currently explored edges. Thus a bound on the total number of edge traversals taken in the first $n - 1$ iterations is

$$\sum_{n_s=1}^{n-1} [n_s + 2(n_s - 1)(2m - 2(n_s - 1))] \tag{3.1}$$

In each of the remaining $m - n + 1$ iterations, one of the hypotheses is accepted. In the worst case scenario the accepted hypothesis is the last hypothesis examined and each traversal comes to the end of the planned path, and all the unexplored (loop) edges are potential hypotheses. The length of each traversal path is bounded by $2(n-1)$, and we can bound the number of hypotheses in each of the remaining iteration i (where $i = 1.....m-n+1$) by $2m - 2(n-1) - 2(i-1)$ which represents all the unexplored loop edges in iteration i, where $i - 1$ represents the number of already explored loop edges in iteration i. Thus, a bound on the total number of edge traversals taken in the remaining $m - n + 1$ iterations is

$$\sum_{i=1}^{m-n+1} [n + 2(n-1)(2m - 2(n-1) - 2(i-1))] \tag{3.2}$$

The total number of steps in the algorithm is bounded by (3.1) + (3.2), which simplifies to

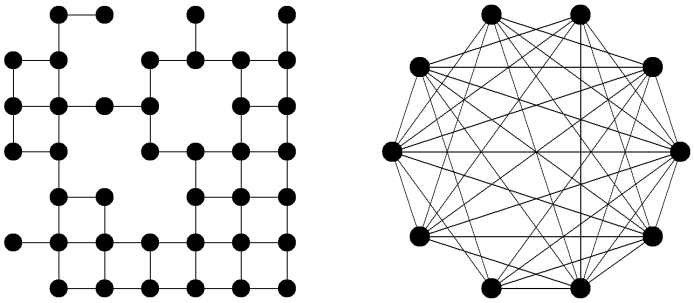
$$2m^2n - 2mn^2 + \frac{2}{3}n^3 - 2m^2 - \frac{5}{2}n^2 + \text{lower order terms} \tag{3.3}$$

So the asymptotic complexity of the algorithm is $O(m^2n)$, where m and n are the number of edges and vertices in the world respectively.

3.5.3 Actual performance

The actual performance of the single directional immovable marker algorithms is evaluated by conducting experiments on several classes of graphs using simulation. The algorithm is evaluated on two-dimensional 'lattice hole' graphs, which are lattice graphs with a specified fraction of randomly selected edges removed (Figure 3.8(a)). Such graphs represent environments that are often encountered in the interior of modern buildings. For comparison purposes, the algorithm is also evaluated on homogeneous graphs and densely connected graphs. Specifically, the algorithm is evaluated on homogeneous graphs including lattice graphs and complete graphs, as well as non-homogeneous graphs including lattice hole graphs and complete graphs with a fraction of randomly selected edges removed (Figure 3.8(b)). Results on homogeneous graphs are shown in Figure 3.10(a)) and Figure 3.10(b). Results for non-homogeneous graphs are shown in Figure 3.10(c) and Figure 3.10(d). For the non-homogeneous graphs average costs are reported for 30 random graphs with 10% of edges or nodes removed. The lower bound m and the upper cost bound derived in expression (3.3) – ignoring the lower order terms – are also plotted.

The algorithm is also evaluated on 'small-world' graphs. Watts and Strogatz [68] define a small-world graph as a one-dimensional lattice with periodic boundary conditions, and a small number of shortcuts bonds added between randomly chosen pairs of sites. We create a n nodes small-world graph by first creating a ring over n nodes, where each node in the ring is connected

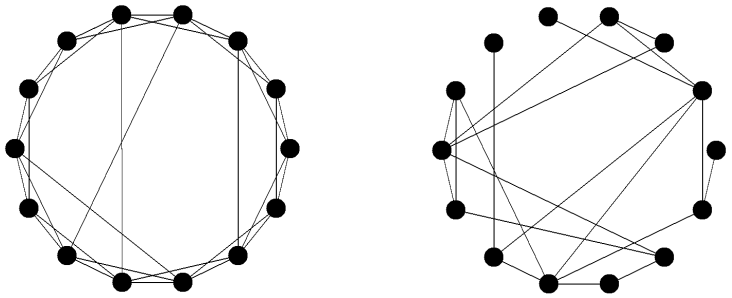


(a) (b)

Figure 3.8: Lattice graph with a fraction of edges removed and densely connected graph with a fraction of edges removed.

with its k nearest neighbors. Then shortcuts are created by adding new edges as follows: for each edge (u, v) with probability p add a new edge (u, w) with randomly-chosen existing node w. An example of small-world graph is shown in Figure 3.9(a). Results for small-world graphs of different k and p are shown in Figure 3.11. We also examine the algorithm on randomly connected graphs, which is created by randomly connecting a specified fraction of edges (over all the possible edges). An example is shown in Figure 3.9(b). Results for randomly connected graphs of different fractions of edge connections is shown in Figure 3.12.

Results for all the examined classes of graphs show that the performance cost for each sized graph is substantially below its theoretical upper cost bound. Meanwhile, the cost is significantly above the theoretical lower bound for mapping and exploration. Motivated by the discrepancy, we present some refinements to the basic single directional marker algorithm later in this work.



(a) Small-world graph with range $k = 2$, and new edge (short-cut) adding probability $p = 10\%$.

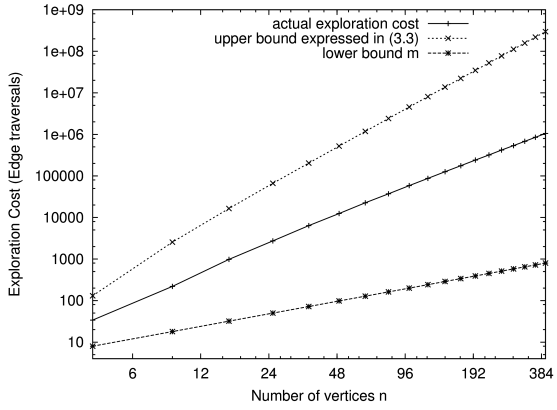
(b) Random graph with 20% edge randomly connected

Figure 3.9: Small-world graph and randomly connected graphs.

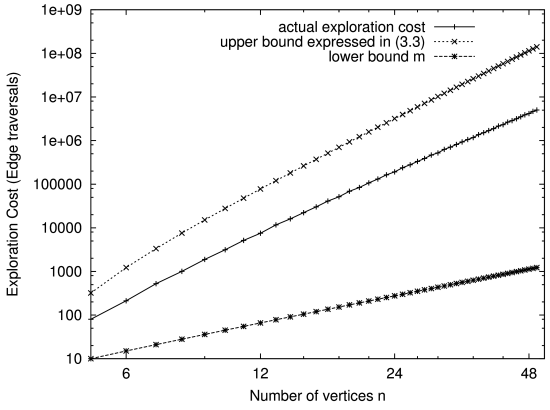
3.6 Physical implementation of the single directional marker algorithm

The single directional immovable marker algorithm described above is a provably correct solution to topological SLAM. This algorithm and other marker-based deterministic topological SLAM algorithms in the literature assume solutions to a number of critical problems in sensing and locomotion, including proper transition of edges, proper characterization of vertices and enumeration of edges in vertices. A critical question when evaluating these algorithms is "are these perception and motion commands realistic when applied to real world environments, sensors and robotic platforms". This section presents the implementation of the single directional immovable marker algorithm using a real robot system[11]. This implementation seeks to provide a constructive answer to this question. This implementation also enables evaluation of other assumptions given for the algorithms. For example, the assumption that the cost of physical motion by the robot related to the cost associated with the computational effort.

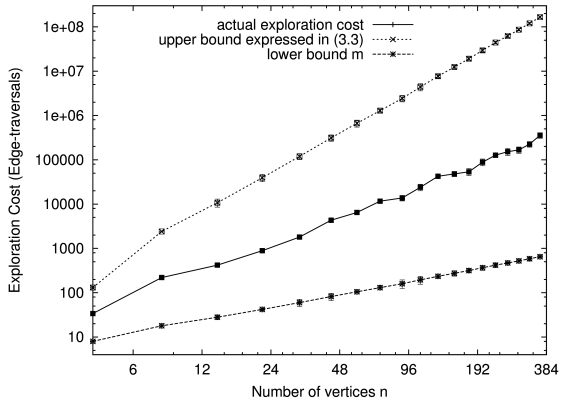
[11]This work has been published [67].



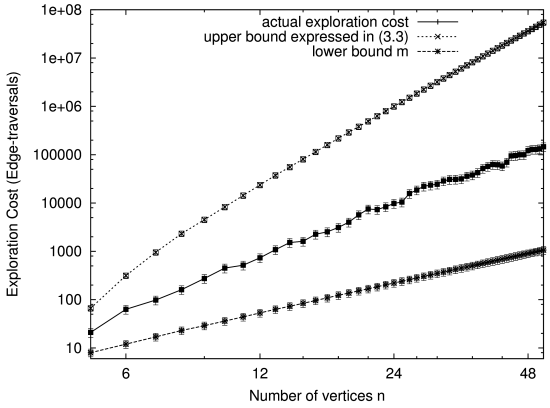
(a) Homogeneous lattices of varying sizes



(b) Homogeneous complete graphs of varying sizes

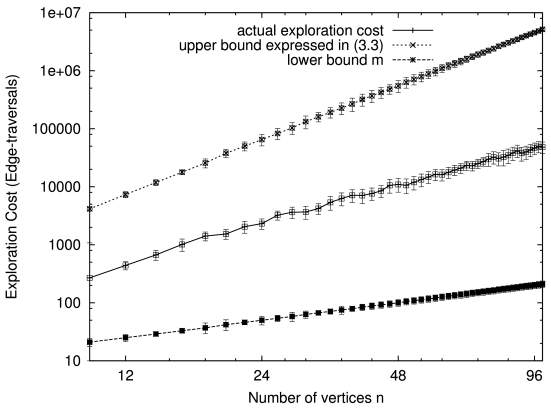


(c) Lattice graphs with 10% randomly removed edges. Varying sizes.

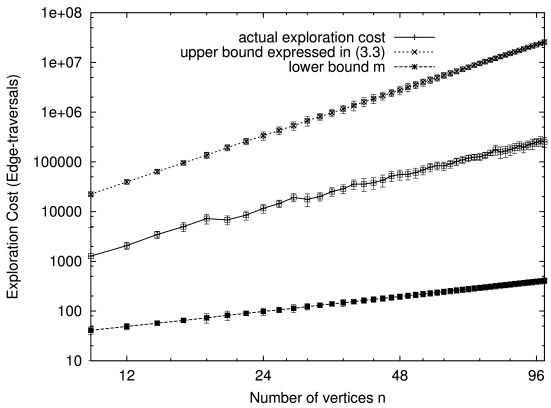


(d) Densely connected graphs (complete graphs) with 10% randomly removed edges. Varying sizes.

Figure 3.10: Performance of the single immovable directional marker algorithm on different graph types. Results in (c) and (d) are averaged over 30 graphs, each with randomly removed edges. Error bars in (c) and (d) show standard deviations.

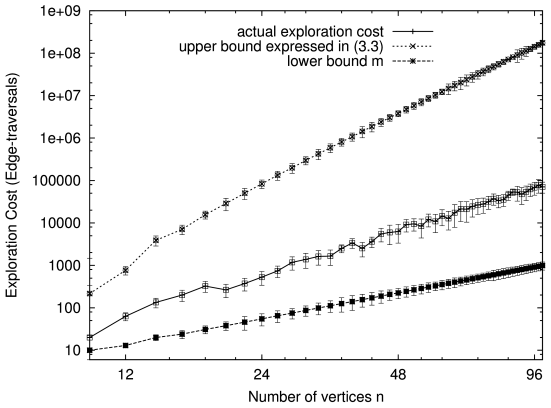


(a) Small-world graphs of range $k = 2$ and $p = 10\%$.

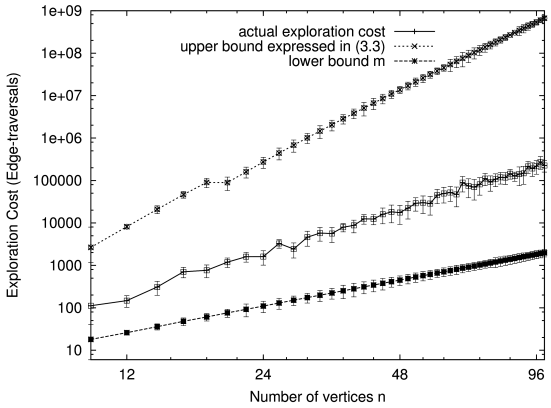


(b) Small-world graphs of range $k = 4$ and $p = 20\%$.

Figure 3.11: Performance of the single immovable directional marker algorithm on small-world graphs. Results are averaged over 30 graphs. Error bars show standard deviations.



(a) Random graph with 20% edge connections



(b) Random graph with 40% edge connections

Figure 3.12: Performance of the single immovable directional marker algorithm on randomly connected graphs. Results are averaged over 30 graphs. Error bars show standard deviations.

(a) original super-scout robots in VGR lab.

(b) super-scout robot augmented with an omni-directional vision sensor.

(c) spherical mirror of the vision sensor.

Figure 3.13: Real robot system established for experimental validations. An original super-scout robot in the VGR lab is augmented with omni-directional vision sensors, which is composed of a digital camera and a spherical mirror.

The environment Different solutions for sensing and mobility are required for different environments. Here the implementation is designed to operate in our laboratory building which is a traditional hallway environment found in many universities and the sensing/mobility solution described here is tuned to this environment. In order to enable us to construct a range of different environments within the building, we have constructed the sensing module of the algorithm to respond to walls marked by tape of a contrasting color. The tape mimics the contrasting wall stripe that marks the wall-floor boundary in the environment. Although the use of tape to mimic the contrasting wall color removes a bit of the reality of the resulting experiments, this modification enables the simulation of a range of different worlds within our building. We model the environment as a graph-like world where hallway junctions correspond to vertices in the topological world, and hallways correspond to edges. Following the layout of our building, we assume that the environment contains three types of junctions: 1-way junctions (dead end), L shaped 2-way junctions and T shaped 3-way junctions (intersections). All intersections between

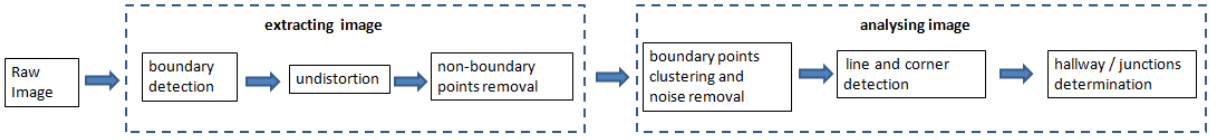


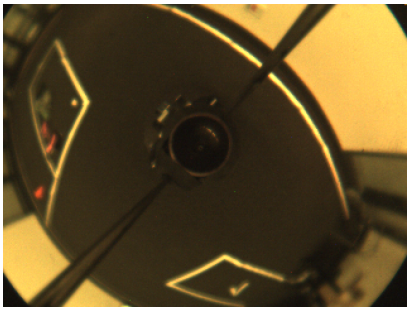
Figure 3.14: Architecture of the sensing module.

corridors are at multiples of 90 degrees. There is a flooring edge between the floor and walls throughout the building created by placing white or black tape on the ground (Figure 3.15(a)).

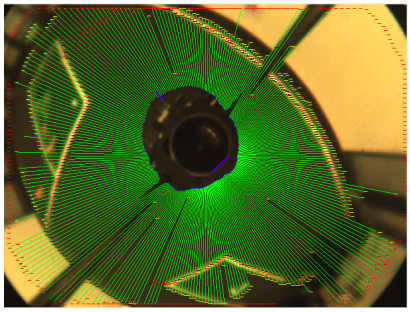
Hardware setup The single directional immovable marker algorithm was implemented using a modified Nomad SuperSout robot. The SuperScout robot is based on a differential drive and is equipped with a ring of sonar sensors. The original robot has been updated a number of times (see [13]) and exposes basic operations through standard ROS [47] nodes. The robot is augmented with an omni-directional video system. (Figure 3.13). The video system is composed of a color USB camera connected to the computer mounted on the robot, and a sphere-shape mirror on the top of the camera. The height of the sensor was chosen to ensure that the wall and floor structure is well captured by the sensor. The sensor, once triggered, produces a panoramic image of the environment that the robot is currently in.

3.6.1 Sensing

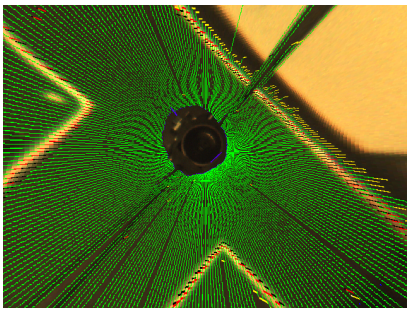
The single directional immovable marker algorithm and other algorithms developed in this thesis require a sensing module that can determine if the robot is in an edge or a vertex. When in an edge this sensing module must be able to aid in navigation to the adjacent vertex, and when in a vertex the sensing module must be able to enumerate the number of edges and be able to inform



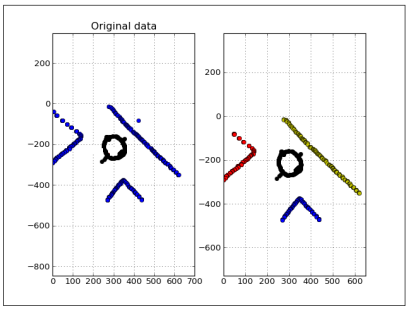
(a) Original panoramic image.



(b) Boundary points detected.



(c) Orthographic view.

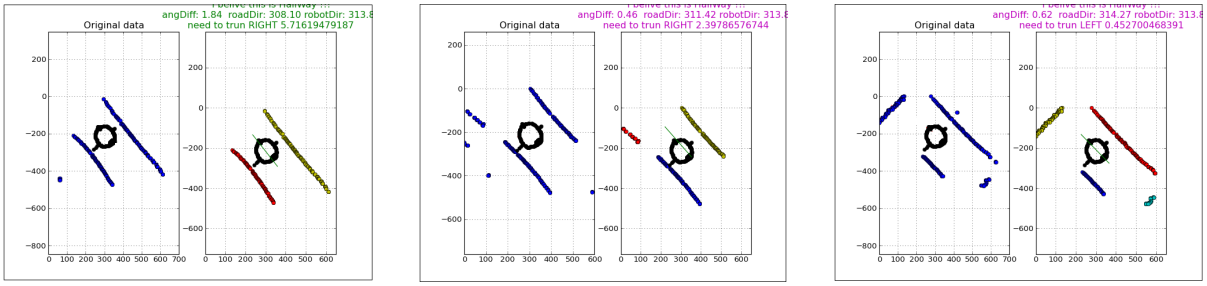


(d) Hallway wall detection and classification.

Figure 3.15: Sensing module operation. The original image (a) is first classified in color space to obtain floor and wall segments (b). This image is then warped so that the ground plane provides an orthographic view of the environment (c) from which wall locations are extracted and classified (d).

the navigation module so as to allow the robot to exit the vertex along the desired edge. The steps in the sensing module are summarized in Figure 3.14.

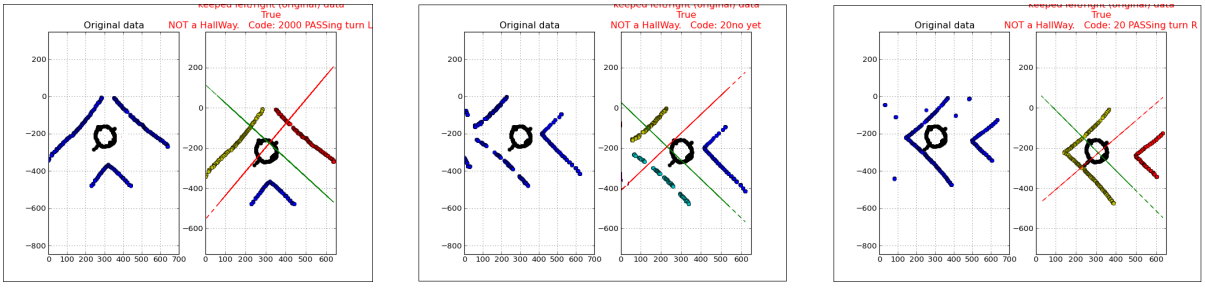
The first step of the sensing module is to detect the boundaries between the floor and walls. A Bayesian classifier is used to map image color represented in HSV space to floor/boundary classes. A boundary determination process identifies wall-floor boundaries in a local frame around the robot. The resulting image is projected to an orthographic view centered on the robot and non-boundary points discarded. This process is illustrated in Figure 3.15.



(a) Ideal case. (b) Multiple wall segments. (c) Obstructed hallway.

Figure 3.16: Various situations in a hallway. The green lines indicate the direction of the hallway.

Pixels identified as corresponding to wall locations are clustered and grouped into wall sections. Small sized clusters are discarded as they typically correspond to noise responses from the pixel classification process. The local environment about the robot is then characterized in terms of the needs of the SLAM algorithm. We first determine if the robot is in a hallway. If only two clusters exist, then we can detect a hallway by checking whether both the clusters correspond to straight lines, are parallel to each other, and are on the opposite sides of the robot. Depending on the position of the robot, there exist several situations where a hallway can result in more than two clusters. For example, there may exist multiple lines on one or both sides of the robot, or there may exist other clusters behind or in front of the robot. Some examples of hallways are shown in Figure 3.16. We detect the two 'major' clusters which are line clusters that are closest to the robot. Principal component analysis (PCA) is used to estimate the orientation of each of the two major clusters, and the orientation is used to check if the clusters are parallel. Using the cross product, we also determine if the major clusters are on opposite sides of the robot. We also determine the status of the hallway. The hallway is 'clear' if only the two major clusters exist. If additional clusters exist behind the robot, they are ignored and we also treat the hallway as

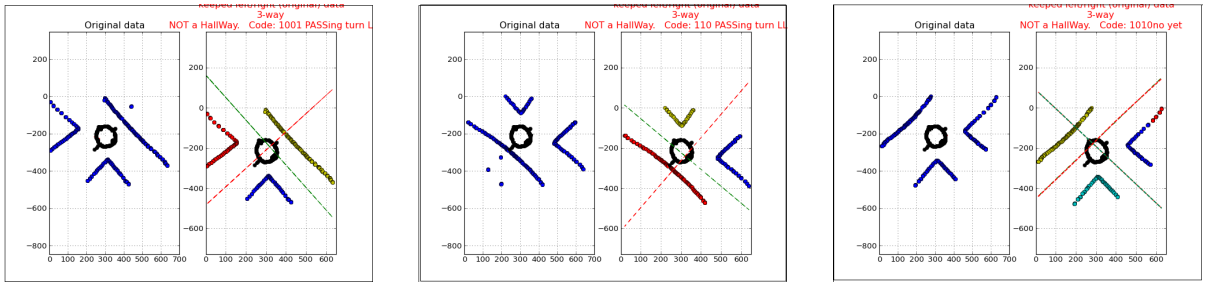


(a) (b) (c)

Figure 3.17: Various situations in a two way junction. The green and red lines indicate the direction of the edges. In (a)(b) corners are formed by either one or two clusters whereas in (c) each corner exists in one cluster. Clusters are distinguished by colors.

'clear'. If there exist clusters ahead of the robot, an 'unclear' message is produced. We also produce an 'unclear' message when the size difference of the two major clusters is above a threshold (e.g., size/length of one major cluster is only half of the other), or, if one cluster is behind the robot. These scenarios are typical indications that the robot is approaching a junction. These warning messages are used to drive the robot in a more careful manner. This step also computes and returns the distance of the robot to the two major clusters. This information is used in later steps to help the robot navigate through hallways.

If no hallway is detected, then the input clusters are further examined to see if they correspond to a junction (vertex). We do this by detecting corners in the wall structure and then determining the number, direction of corners and their relative positions. Depending on the position of the robot in a junction, a corner may exist in a single cluster or may be formed by two or more clusters, as shown in Figure 3.17. We consider both possibilities. Two clusters form a corner if both the clusters are well modelled by straight lines, they are perpendicular to each other, and in addition, one end of a cluster is close enough to one end of the other cluster (the two ends



(a) (b) (c)

Figure 3.18: Various situations in a 3-way junction. The green and red lines indicate the direction of the edges. Each corner exists in one cluster.

'touch'). If they form a corner, then depending on the two 'touching' ends of the clusters, the corners are classified into different 'types', which represent the turning direction of the corners. The type of corners is used to derive the type of a junction in the next step. Corners formed by multiple clusters can be found in Figure 3.17(a) and 3.17(b). After detecting corners formed by multiple clusters, we further consider the possibility that a cluster itself may contain corners, which usually happens when the robot is closer to the corner. In order to detect possible corners in a single cluster, we first try to decompose each cluster into line segments, and then detect possible corners formed by the segments in a similar manner as above. Corners formed by a single cluster can be found in Figure 3.17 and 3.18.

After detecting all the possible corners in the boundary points, we proceed to determine if a junction has been formed by the corners, based on the type and relative position of the corners and possible lines. For example, a two way junction should be formed by two corners of the same type, whereas a three way junction is formed by two corners of different types and a straight line. Moreover, based on the type of the corners we can figure out which edge the robot is entering,

and thus which direction to turn for each other edge of the junction. Once a junction is detected, we compute midpoints of the edges of the junction, as shown in Figure 3.17 and 3.18. Here, the green lines indicate the direction of the edge the robot is traversing, and the red lines indicate the directions of the other corridors to follow upon exiting this junction. These lines identify the various trajectories that the robot can follow through the vertex/junction. Specifically, the central lines serve two purposes: 1) based on the angle of the lines and robot, later stages can drive the robot along the current edge (represented by the green lines in the figure) or turn to other edge (the robot can enter a junction in different angles so we cannot always assume a 90 degree turn – the robot needs to turn according to the angle difference with the red line in the figure). 2) by examining if the robot has passed the red line (based on the cross product of the robot to the red lines), we can determine if it is time to turn or if the robot has just entered the junction so it is not yet time to turn.

3.6.2 Motion

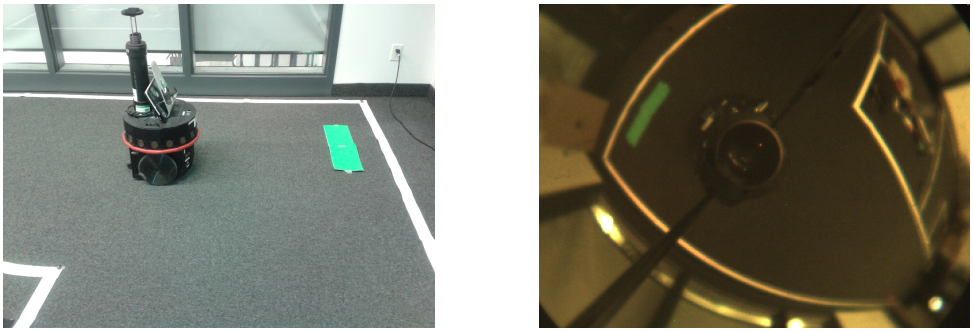
Based on the information returned from the sensing module, we implement the fundamental motion capabilities of the robot: executing a motion sequence. That is, given a sequence of relative edges to follow, traversing the environment autonomously by executing the relative door specified at each visited vertex. The robot should follow hallways and make movement at each junction by traversing the corresponding relative exit at the junction.

Traversing in hallways. If the robot is in a 'clear' hallway, then the robot moves forward for a pre-defined unit distance (e.g., 50 cm). On the other hand if the hallway is 'unclear', then the

robot moves forward a smaller distance (e.g., 25 cm). In both cases, before making the forward move, based on the direction of the hallway and the robot's direction, the robot adjusts its orientation to align with the direction of the hallway. Moreover, based on the returned distance to the major clusters, we detect if the robot is too close to one side of the hallway, and if it is, the robot moves closer to the center of the hallway.

Navigating junctions. For a junction, the sensing module returns the type of junction (1/2/3 way), the central lines of the edges, and whether the robot has reached or passed the red line. If the robot has not yet reach the planned turning point, then the robot adjusts its orientation by aligning itself toward the intersection of the green line and the red line, and then moves to the line intersection, ending up at the center of the junction. If the robot has reached or passed the red line, then the robot turns towards the planned exit door based on the angle to the red line, followed by a forward move (to leave the junction).

Path execution. The Sensing and Motion modules described above are combined into a Path Execution module. This module takes as input a path to be executed (represented as a sequence of relative edges), and provides an implementation of the sensing and motion required to enable the robot to navigate a hallway-like environment without resorting to global metric information. This module is sketched in Algorithm 3.2.



(a) (b)

Figure 3.19: A directional immovable marker in a 2-way junction.

3.6.3 Detecting a directional marker in a junction.

The final remaining sensing requirement in order to implement the single immovable directional marker algorithm is to develop a directional immovable marker and corresponding sensing algorithm. To simplify this, we assume that the marked place is a two-way junction and we mark the floor of the junction with a rectangular object aligned with one of the edges (Figure 5.1). The marker is constructed using a unique color which makes it easily detected when the robot is in the junction. The encoding and detection of the direction indicated by the junction are also straightforward. Specifically, given an image of the environment, the existence of the marker is determined by the occurrence of large amount of unique color pixels, and the direction of the marker can be determined using PCA on the marker pixels.

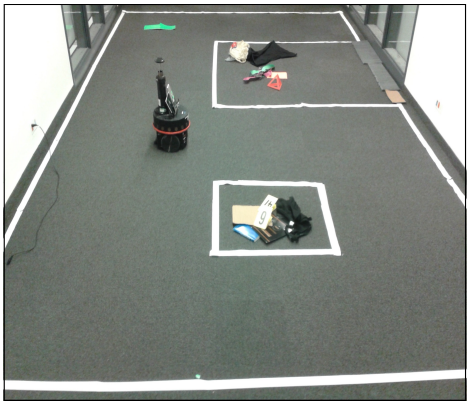
3.6.4 Experiments

Experiments have been conducted both in indoor simulated environments where a white flooring edge is used, as well as on the main floor of the real building. In the simulated environments the

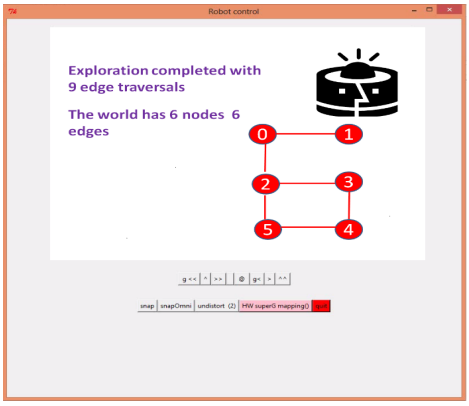
Algorithm 3.2: Path execution

Input: a queue Q of relative edges to follow at junctions

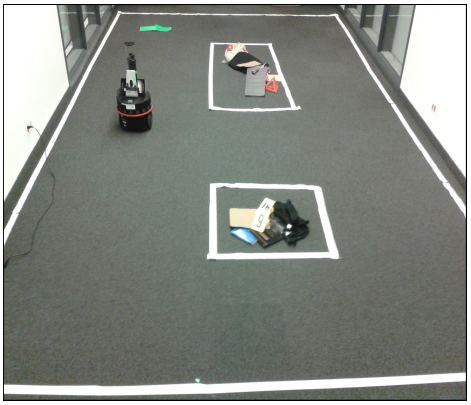
```
1  state ← LEAVING; d ← a unit distance;
2  while true do
3      take a panoramic picture;
4      send the picture to the sensing module;
5      type, status = output of the sensing module;
6      if type == hallway then
7          state ← HALLWAY;
8          adjust direction to align with hallway direction;
9          move toward center line if too close to one side;
10         if status == clear then
11             move forward distance d;
12         else // status == unclear;
13             move forward distance d/2;
14     else if type == junction then
15         if status == not yet red line then
16             state ← JUNCTION;
17             adjust direction toward the intersection of green line and red line;
18             move to the line intersection; // end up at the junction center;
19         else // status == passing red line;
20             if Q is empty then
21                 break; // finish the path, stop;
22             door ← a relative edge removed from Q;
23             steps ← steps + 1; // complete an edge traversal;
24             turn left, right, go ahead, or turn back based on junction type and door;
25             move forward distance d; // leave the junction;
26             state ← LEAVING;
27     else if type == unknown then
28         if state == HALLWAY or LEAVING then
29             move forward distance d/3; // careful;
30         else // state==JUNCTION or UNKNOWN;
31             turn a little angle to one side; // dangerous to move forward;
32         state ← UNKNOWN;
```



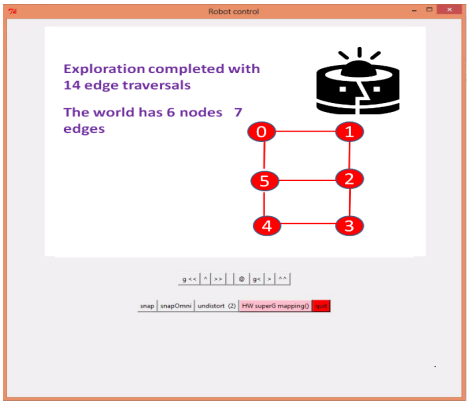
(a) Exploring a 'shape 6' world.



(b) Generated map for the world in (a).



(c) Exploring a 'shape 8' world.



(d) Generated map for the world in (c).

Figure 3.20: Exploring different indoor environments. Snapshots of recovered maps are shown in (b) and (d).

algorithm has been evaluated on a number of different graph-like worlds. The robot mapped the worlds successfully. Some results are shown in Figure 3.20. The algorithm has also been tested successfully on a subset of the main floor of our building under natural and varying lighting conditions[12].

In order to examine the cost associated with physically moving a robot and the cost associated

[12]Some videos for the experiments can be found at http://vgrserver.cse.yorku.ca/~huiwang/Sites/thesis.html.

with the computational effort, an off-line version of the algorithm is implemented. Instead of driving the robot to do exploration, the off-line algorithm takes as input the panoramic images taken in the above experiments and does not involve real motion by the robot. The off-line algorithm was run on the same set of simulated environments (using the already generated panoramic images for the environment), and the time taken is compared with that taken by the algorithm running on the real robot. The running time required by the off-line algorithm includes all the computation effort including image processing in the sensing module. For the shape 6 world shown in Figure 3.20, the real robot takes the robot about 25 minutes to complete exploration and for the shape 8 world it takes about 35 minutes to complete[13]. When running the off-line algorithm using the same computer hardware, no more than 5 minutes are used for both environments. Similar results are obtained for other environments. So in the real exploration, a small fraction of time is spent on the computational cost. The bulk of the time is spent on physically moving the robot.

3.7 Summary and discussion

Given an embedded topological world, it is not, in general, possible to map the world deterministically without resorting to the use of sufficient place and back-link information to solve the 'have I been here before?' problem. This chapter shows that neither can knowledge of a vertex providing explicit place information or explicit back-link information alone enable the robot to always solve the problem deterministically. This chapter also shows that given a single 'directional

[13]Computation took place on a Lenovo Ideapad computer with Intel(R) Core(TM) i7-3537U processor@ 2.00GHz, 8GB DDR3 memory and 64-bit Operating System.

lighthouse' vertex that provides both explicit place and back-link information, a provably correct mapping strategy using such information exists with cost bound $O(m^2n)$. Earlier algorithms required the robot to manipulate markers in order to map the world. With the development of the directional lighthouse algorithm, it is now possible for the robot to map a topological world deterministically without manipulating markers. Empirical evaluation of the algorithm shows that the real costs on the different environments are much lower than the derived upper bound, and the cost is also significantly above the theoretical lower cost of exploration and mapping.

Most of the deterministic topological mapping algorithms have only been implemented in simulation and it has been an open question as to how well the assumptions of these algorithms would transfer to real robot. By implementing the single directional lighthouse on a real robot, this chapter provides a constructive answer to this question. The implementation here uses very simple sensing coupled with straightforward local path planning to implement a deterministic SLAM algorithm. More sophisticated sensing and planning would be required for more complex environments. The development of such modules is the topic of future research.

We conclude this chapter by discussing different ways that a 'directional lighthouse' might be established. Such information can be established at a vertex by marking the vertex with one directional immovable marker, which is considered minimum marker in terms of the number of markers and the movability of markers. Such information can also be established with a cluster of undirected immovable markers. This chapter showed that one undirected immovable vertex marker does not necessarily establish both explicit place and back-link information. It can also be shown that two undirected immovable vertex markers are not necessarily sufficient to

establish such information either. An example is shown in Figure 3.21(a). Upon entering one of the marked vertices, say, v_0, by traversing edges looking for the marked neighbor, the robot can determine its back-link at v_0 (i.e., by which edge it enters). Thus explicit back-link information exists in the v_0 (and the other marked vertex) and but place information at the marked vertices is ambiguous – the robot cannot determine which marked vertex it is visiting. It can be justified (with induction) that if the robot executes identical motion sequences, starting from v_0, then the acquired perception on the different graphs are the same. (The proof is similar to the proof given for the single undirected marker case and is omitted here.) Thus the robot cannot always map an arbitrary world deterministically with explicit back-link information but no explicit place information. It can also be shown that three undirected markers, one in a vertex and two in a neighbor vertex do establish both explicit place information and explicit back-link information in the marked vertices, as shown in Figure 3.21(b). Upon entering a marked vertex, the marked place can be identified based on the different marker count, and the entry edge can be established by taking extra traversals looking for the marked neighbors.

While one or two undirected immovable vertex markers are not necessarily sufficient to establish a directional lighthouse, they can be sufficient to map deterministically a graph that contains unique structure that, once marked, provides direction and localization information and thus be used as a directional lighthouse. One example of such a structure is a vertex of degree one. Such a vertex, once marked with an undirected marker, forms a unique directional signature that can serve as a directional lighthouse. By expanding the local signature, we can generalize this type of directional lighthouse. For example, by extending the signature of a vertex with signatures of

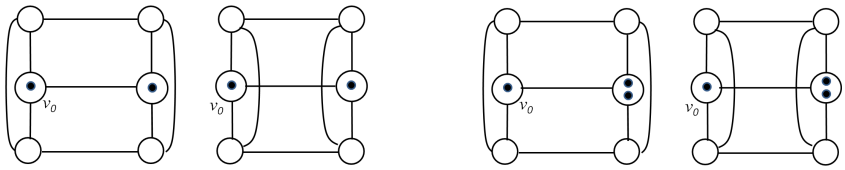


Figure 3.21: (a) Two different embedded graphs that are not distinguishable with two undirected markers. Assume a clockwise enumeration convention, and that the robot starts from v_0. $\mathcal{M} = \{1, 1, 2, 2, 1\}$ results in same perception $\mathcal{P} = \{$[3,A,A][3,A,A][3,A,A][3,A,A][3,V-1,A]$\}$ on both the graphs, where [3,V-1,A] means that the marked neighbor is present and is the next on the left. (b) Three undirected markers establish explicit place and back-link information at v_0. Motion $\mathcal{M} = \{1, 1, 2, 2, 1\}$ results in different perceptions $\{$[3,A,A][3,A,A][3,A,A][3,A,A][3,V-1,A]$\}$ and $\{$[3,A,A][3,A,A][3,A,A][3,A,A][3,V-2,A]$\}$ on the graphs.

its immediate neighbors, the undirected immovable marker algorithm can be used to establish a directional lighthouse in a vertex which has a neighbor vertex whose degree is different from all the other neighbors. (A vertex with degree one is a special instance of this general structure.) This concept is investigated further in Chapter 5. The directional lighthouse can also be established in other ways. This is discussed in Chapter 6.

Chapter 4

Enhancements to the single directional immovable marker algorithm

Given the potentially high cost of the basic single directional lighthouse algorithm given in Chapter 3, this chapter explores enhancements that can be made to the basic algorithm so that the exploration cost is reduced. The high cost of the single directional immovable marker algorithm and its variations comes from the motion-based validation step required to validate or reject each hypothesis. There are many ways to reduce the exploration cost, and this chapter explores two classes of approaches. These approaches try to reduce the validation cost by reducing the number of potential hypotheses, by reducing the number of hypotheses that need to be validated with motion, and by reducing the amount of motion required to validate an individual hypothesis. The first class of enhancements reduce the validation cost by exploiting the traversals required to reject a hypothesis. When a hypothesis is rejected, these approaches utilize the traversals taken to reject this hypothesis to examine other yet unvalidated hypotheses, in an attempt to reject other hypotheses without motion, or to reduce the amount of motion required to validate the remaining hypotheses. By constructing expanded local signatures of places using extra traversals,

the second class of approaches attempt to reduce the number of potential hypotheses. Note that these enhancements do not improve the theoretical exploration cost bound $O(m^2n)$ of the algorithm; rather they are optimizations that find substantive improvements in practice. Portions of this chapter have appeared in [66].

4.1 Exploiting traversals of validated hypotheses

In the basic single directional immovable marker algorithm, each hypothesis is validated with motion, and validated independently. When the robot enters the unknown end vertex v_u via an newly explored edge e, each hypothesis h' for e and v_u is validated by executing its motion sequence $\mathcal{M}_{h'}$. If a mismatch between the computed (expected) perception $\mathcal{P}^E_{h'}$ and the sensed perception $\mathcal{P}_{h'}$ is detected during motion execution, then hypothesis h' is rejected. In this case the robot terminates validation and executes the reverse of $\mathcal{M}_{h'}$, returning back to v_u, and then executes the motion sequence for a yet to be validated hypothesis h'' (if any). This class of enhancements tries to reduce the effort spent conducting traversals by exploiting the traversals of a rejected hypothesis. Three approaches are discussed below. These approaches require no addition of motion by the robot but require increased computational effort and memory.

Enhancement-1: Capturing overlap of execution paths

For each hypothesis $h' = (e', v_{k'})$, the computed motion sequence $\mathcal{M}_{h'}$ would drive the robot to v_0 via the expected entry edge if the robot is at the hypothsized place $v_{k'}$ with the hyposthezied back-link e'. That is, the actual path traversed by the robot in executing $\mathcal{M}_{h'}$ matches the

planned path if and only if h' is true. Call the actual path (traversed or would be traversed) in executing $\mathcal{M}_{h'}$ the *execution path* of h'. The key observation here is that for an newly explored edge e and v_u, the execution paths of the hypotheses may partially or even completely *overlap*. That is, the motion sequences for the hypotheses, once executed, may lead to traversals on same edges and vertices. One kind of exection path overlap is captured: the execution paths overlap from the beginning (vertex v_u). While differenet motion seqences may assume different hypothesized starting places and back-links, the actual execution of the motion sequences all start from the same place v_u and entry edge e. Thus while it is not necessarily known where each executed path visits until the world is fully explored, it is known that beginning from v_u the same motion sequence (relative doors) visits the same edges and vertices in the environment. For example, motion sequences (2,1,3) and (2,1,4) for two hypotheses define execution paths that overlap at the first two edges and vertices, excluding the starting vertex v_u. As another example, we have seen in Figure 3.7 that hypothesis $h' = (e', v_{k'})$ and $h''' = (e''', v_{k''})$ have exactly the same motion sequence (2,2,1), which define execution paths that overlap completely. Suppose that the robot, starting from v_u, executes a motion sequence of a hypothesis h' and stops after it has executed (2,1,3,1,2) when a mismatch is detected. Also suppose now a yet-to-be validated hypothesis h'' has motion sequence $\mathcal{M}_{h''} = (2, 1, 3, 3, 2, 4)$, which defines an execution path that overlaps with the executed path of h' at the first three edges and vertices, as shown in Figure 4.1. We can exploit this overlap to reduce the number of edge traversals that are required to validate the second hypothesis. This enhancement exploits the overlap in two stages. The first stage tries to reduce the number of hypotheses that require motion, and the second stage tries to reduce the

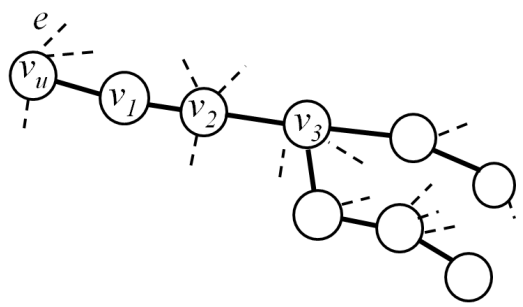


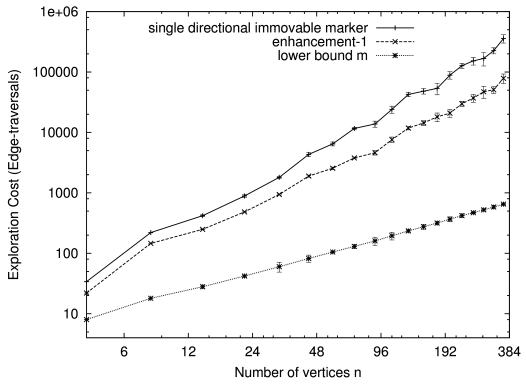
Figure 4.1: Overlap of motion sequences (2,1,3,1,2) and (2,1,3,3,2,4) of two hypotheses. Overlap at the first three edges and vertices visited (excluding v_u). Note that the two hypotheses incident on different known verteices.

motion required to validate a hypothesis. First, the overlap due to the first three motions implies that if the robot executes motion sequence $\mathcal{M}_{h''}$ of h'' from v_u (as in the original algorithm) and finishes, it would visit exactly the same first three vertices (v_1, v_2 and v_3) and edges in the executed path of h'. The observed perception for h'' at the first three vertices (v_1, v_2 and v_3), denoted $\mathcal{P}_{h''}^{overlap}$, must be the same as the observed perception $\mathcal{P}_{h'}^{overlap}$ for h' in the first three vertices, i.e., $\mathcal{P}_{h'}^{overlap} = \mathcal{P}_{h''}^{overlap}$. Since h'' is rejected if a mismatch between the observed perception $\mathcal{P}_{h''}$ and its expected perception $\mathcal{P}_{h''}^{E}$ is detected at any vertex along the path, it is clear that h'' can be rejected without executing its motions $\mathcal{M}_{h''}$ if the observed perception $\mathcal{P}_{h'}^{overlap}$ for h' at the first three vertices does not match the expected perception $\mathcal{P}_{h''}^{E-overlap}$ of h'' at the first three vertices. Thus, in this example when the robot stops its motions for h' due to a signature mismatch, it remains at the vertex where it stopped (instead of coming back to v_u as in the original algorithm, as is required in the second stage of this enhancement as discussed next), and compares the perceptions $\mathcal{P}_{h'}^{overlap}$ the robot just obtained at the first three vertices against the expected perception $\mathcal{P}_{h''}^{E-overlap}$ of h'' at the three vertices. If $\mathcal{P}_{h'}^{overlap}$ and $\mathcal{P}_{h''}^{E-overlap}$ do not

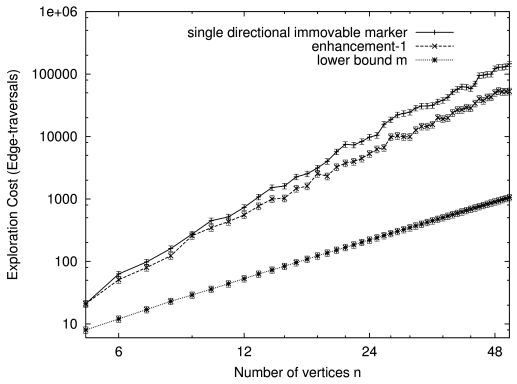
match exactly, hypothesis h'' is rejected immediately without executing its motion sequence.

What if $\mathcal{P}_{h'}^{overlap}$ and $\mathcal{P}_{h''}^{E-overlap}$ match? In such a case, the second stage of the enhancement starts, in which we apply another technique to exploit the overlap, trying to reduce the motion required to validate a hypothesis. The key observation here is that now the robot needs to validate h'' with motion, but since the path just traversed for h' and the execution path of h'' overlap, the robot only needs to validate (traverse) the *non-overlapped* part of the execution path for h''. To do this, the robot retraces its motions but only until it comes back to the *first* overlapped vertex it encounters on the way back (v_3 in the example). Then the robot continues to execute the non-overlapped part of the motion sequence of h''. In this example, the robot executes (0, -2, -1) with 0 and -2 leading the robot back to v_3 and -1 re-orientating the robot to original entry edge at v_3, and then attempts to execute (3, 2, 4). Now further suppose that the robot detects a mismatch after executing non-overlapping motions 3 and 2. The robot then repeats the above process by staying at the current location and examining the possible path overlap with another unvalidated hypothesis h''' (if any). Note that in calculating the possible path overlap of h'' and h''', the algorithm uses the overlapped motion (2,1,3) *and* the executed non-overlapping motion (3,2) as its executed motion (2,1,3,3,2), i.e., $\mathcal{M}_{h''} = \mathcal{M}_{h'}^{overlap} + \mathcal{M}_{h''}^{non-overlap}$. Correspondingly, the algorithm uses the observed perception for e' at the overlapped vertices *and* the observed perception for e'' at the non-overlapped part as the observed perception for the whole sequence for e'', i.e., $\mathcal{P}_{h''} = \mathcal{P}_{h'}^{overlap} + \mathcal{P}_{h''}^{non-overlap}$. That is, the robot proceeds as if it has traversed from v_u as in the original algorithm. By returning to the first overlapped vertex and continuing on the non-overlapping part of e'', twice traversals on each overlapped edge are saved.

In order to keep the robot 'hanging around' as much as possible before it has to return to v_u, we adopt a *greedy* approach to exploiting the possible overlaps of the paths. When the robot enters an unknown vertex v_u via e, we pre-process all the hypotheses for e, computing the motion sequence $\mathcal{M}_{h'}$ and the expected perception $\mathcal{P}^E_{h'}$ for each loop closing hypothesis h'. One of the hypotheses is chosen as the first hypothesis which is validated with traversals (as in the original algorithm). From then on, whenever the robot stops executing the motion sequence of a hypothesis due to perception mismatch, it remains there and computes possible overlaps between the just executed path and the execution path defined by the motion sequence of each remaining hypothesis. We first compare the perceptions based on the observed perception at the overlapped vertices and filter out (reject) all the hypotheses whose expected perceptions do not match the observed perception at the overlapped places (stage 1). For the remaining hypotheses, whose motion sequences need to be executed, the robot selects the one with the *longest* overlapped execution path segments (i.e., highest number of overlapped vertices), and executes the motion sequence of the selected hypothesis (by coming back to the nearest overlapped vertex and then continues on the non-overlapped part of the hypothesis (stage 2)). Note that in the special case that the execution path of an unvalidated hypothesis is part of a traversed path of the validated hypothesis from the beginning (e.g., the executed motion sequence is (3,2,1,4) and the to-be-executed motion is (3,2,1)), then the robot has already traversed the execution path of the unvalidated hypothesis and thus no movements are required for the unvalidated hypothesis. Also note that if no overlap can be captured from the remaining hypotheses, then the robot moves back to v_u to conduct the other validation there, as is the case in the original algorithm.



(a) Lattices with 10% edges removed



(b) Densely connected graphs with 10% edges removed

Figure 4.2: Performance of enhancement-1 (Capturing overlap of execution paths) on different graphs (log scale). Results are averaged over 30 graphs, each with randomly removed edges. Error bars show standard deviations.

The algorithm is sketched in Algorithm 4.1, where line 16-19 corresponds stage-1 and line 22-23 corresponds to stage-2 of the enhancement.

In order to evaluate the performance of this enhancement, experiments were conducted on the same sets of lattice hole graphs and densely connected hole graphs that were used in evaluating the basic algorithm presented in the previous chapter. Both the original algorithm and the enhanced algorithm were run on these graphs, and the average costs are reported in Figure 4.2, where the theoretical lower bounds are also plotted. Results show that for both the graphs, this enhancement gives significant cost reductions. Moreover, the cost reduction increases as the graph size increases. For example, for a size 90 lattice hole graphs, this enhancement reduces the average cost from 13,800 steps in the original algorithm to 4,600 steps – about a 65% reduction. For a size 260 lattice hole graphs the enhancement reduces the average cost from 152,000 steps to

Algorithm 4.1: Enhancement-1: Capturing overlap of execution paths.

```
1  the robot drops the marker at v_0, pointing to an edge;
2  S ← {v_0}; U ← edges in v_0; // initial S & U
3  while U is not empty do
4      remove an unexplored edge e = (v_k, v_u) from U;
5      the robot traverses S to v_k and follows e to v_u;
6      H ← set of loop closing hypotheses for edges in U and their known end vertices which
       have the same signature as v_u;
7      compute motions and expected perceptions for all the hypotheses in H;
8      h' = (e', v_k') ← a hypothesis removed from H;
9      the robot attempts to traverse the computed path v_k', ..., v_0 of e';
10     while True do
11         based on observed perception P_h' along path v.s. the expected perception P^E_h' do
12             case P_h' and P^E_h' match exactly
13                 confirm h', exit inner 'while' loop;
14             case P_h' and P^E_h' do not match
15                 the robot stops and remains at the location;
16                 for each remaining hypothesis h'' in H do
17                     compute the overlap of executed path of h' against the execution path
                       of h'', and compare perceptions P^overlap_h' against P^{E-overlap}_h'';
18                     if P^overlap_h' and P^{E-overlap}_h'' do not match then
19                         remove h'' from H; // reject h'' without motion!
20                 if H is empty then
21                     exit the inner 'while' loop;
22                 h' ← a hypothesis (unexplored edge) removed from H which has longest
                   overlapped execution path segments;
23                 the robot retraces to the nearest overlapping node v_x and then attempts to
                   traverse the non-overlapped part v_x,...,v_0 of h';
24     if a hypothesis is confirmed then
25         do 'loop augmentation' on S;
26     else // no hypothesis exists, or, all hypotheses are rejected
27         do 'non-loop augmentation' on S and U;
28 return S;
```

37,000 steps – about a 75% reduction[14]. Similar results are obtained for small-world graphs and randomly connected graphs. It is interesting to observe the substantial cost reduction for the densely connected graph. For example, for the 32 node graph and the 55 node graphs, average cost reductions of 54% and 64% respectively are observed. Such graphs have a very small diameter due to the dense connectivity, so the to-be-executed paths (to v_0) are usually very short. Due to this, the reduction from each captured overlapping path is small. However, for each e, the number of hypotheses could be large because the degrees of the vertices are large. So the number of overlaps captured could be large. In constructing such graphs, removing edges from complete graphs results in a large number of vertices being affected, so the signature of the vertices are largely distinct. Thus, among the captured overlaps, a large fraction of the overlaps results in hypotheses being rejected without motion (stage-1), while others result in reduced validation motion (stage-2). So the total number of overlaps is large, and thus the total reduction due to the overlap is substantive.

Enhancement-2: Exploiting executed paths that map onto S

The previous enhancement utilizes the traversals of one rejected hypotheses to reduce the exploration cost associated with other hypotheses that have not yet been validated, in the special case of execution paths that overlap from the beginning. This raises an interesting and general question: can we exploit the executed traversals of a rejected hypothesis to examine unvalidated hypotheses, even if the execution paths do not overlap? Suppose that the robot has executed

[14]The fraction of reduction on different sized graphs are summarized in Table 4.1 and Table 4.2 given at the end of the section.

the motion sequence (3,2,1,4) and an unvalidated hypothesis has the motion sequence (1,2,3,4). The execution paths defined by these motion sequences do not overlap from the beginning, and may or may not overlap at some point later (but before e and v_u are validated, we don't know). Here we present another enhancement that extends the earlier approach, exploring techniques that exploit executed motions even if no overlaps are captured.

Suppose that when validating a newly explored edge e, we keep track of all of the executed motion sequences for the rejected hypotheses of e, and their corresponding observed perceptions. Then for an unvalidated hypothesis $h'' = (e'', v_{k''})$ of e, before validating it with motion, we assume for a moment that h'' is true (and thus the robot has entered $v_{k''}$ via e'') and then, starting from this hypothesized place $v_{k''}$ and entry edge e'', trace on the map S *all* the previously executed motion sequences for e. Some of the executed paths map onto known locations on the map S, while others map onto unknown locations. The key observation is that *if* hypothesis h'' is true, then for the previously executed motions that map onto known vertices in the map S, the observed perception for these motions should match the signatures (indicated on the map S) of these known vertices. Thus if any of the perceptions at the mapped known locations do not match the signature of the locations, then hypothesis h'' can be rejected without executing its motion sequence. For example, suppose the robot has executed the motion sequence (3,2,1,2) for hypothesis h', and the corresponding observed perceptions $\mathcal{P}_{h'}$ are ([4,A,A], [2,A,A], [3,A,A], [5,A,A]). Now in validating another hypothesis $h'' = (e'', v_{k''})$, we first assume that h'' is true and then trace the previous motion sequence (3,2,1,2) in the map S, starting from place $v_{k''}$ and entry edge e'' at $v_{k''}$. Suppose that motions 3 and 2 map onto known locations $v_{k'''}$ and $v_{k''''}$

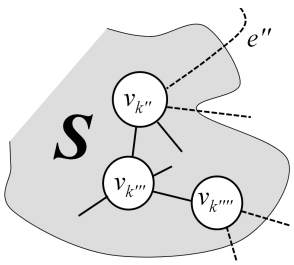


Figure 4.3: Tracing the previous motions on S. Assuming hypothesis $h'' = (e'', v_{k''})$ is true, tracing the motion sequence (3,2,1,2) from $v_{k''}$. Map this sequence onto the known places $v_{k'''}$ and $v_{k''''}$. The signature of $v_{k'''}$ is [4,A,A]. The signature of $v_{k''''}$ is [3,A,A].

in S, and motion 1 visits an unexplored edge, as shown in Figure 4.3. Then we compare the signatures of $v_{k'''}$ and $v_{k''''}$ (indicated in map S) against the corresponding perceptions observed when executing motions 3 and 2 for h'. If the signatures and perceptions do not match, then hypothesis h'' must be wrong and can be rejected immediately without motion. That is, the traversal for h' is used to validate h'' as well. In this example, h'' should be rejected as the signature of $v_{k''''}$ is [3,A,A] but the observed perception is [2,A,A]. If all the perceptions on known locations match the corresponding signatures, then we validate the hypothesis with motions as in the original algorithm, or alternatively, adopt the greedy technique used in the stage-2 of enhancement-1. That is, the robot captures the possible overlap of execution paths, picking the hypothesis having the longest overlapping path segments and then executing the non-overlapped motions. This latter version of the enhancement is sketched in Algorithm 4.2. Note that, as the example shows, in tracing motion sequences, we stop tracing when a motion traverses an unexplored edge (motion 1 in the example) leading to an unknown location as we lack signature information about unknown locations. This issue is addressed in enhancement-3.

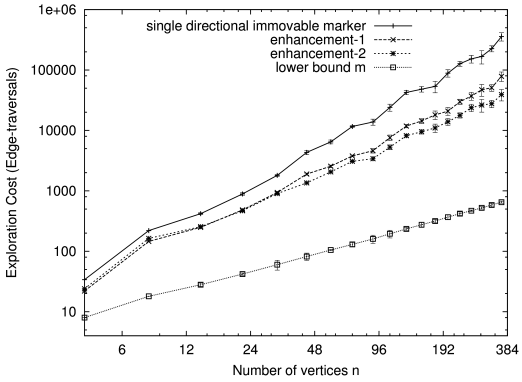
Note that this enhancement incorporates the idea in stage-1 of enhancement-1 (Capturing

Algorithm 4.2: Enhancement-2: Exploiting executed paths that map onto S. Technique in stage-2 of enhancement-1 is used (line 22-24).

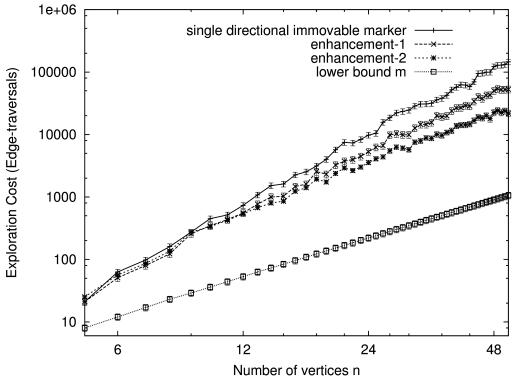
```
1  the robot drops the marker at v_0, pointing to an edge;
2  S ← {v_0}; U ← edges in v_0; // initial S & U
3  while U is not empty do
4      remove an unexplored edge e = (v_k, v_u) from U;
5      the robot traverses S to v_k and follows e to v_u;
6      H ← set of loop closing hypotheses for edges in U and their known end vertices which
       have the same signature as v_u;
7      compute motions and expected perceptions for all the hypotheses in H;
8      h' = (e', v_k') ← a hypothesis removed from H;
9      the robot attempts to traverse the computed path v_k', ..., v_0;
10     while True do
11         based on observed perception P_h' along path v.s. expected perception P^E_h' do
12             case P_h' and P^E_h' match exactly
13                 confirm h', exit inner 'while' loop;
14             case P_h' and P^E_h' do not match
15                 the robot stops and remains at the location;
16                 for each remaining hypothesis h'' in H do
17                     assume h'' is true, trace all the previously executed motions for e;
18                     if a motion mapped onto a known place in S, but the observed
                       perception does not match the signature of the known place on S then
19                         remove h'' from H; // reject h'' without motion!
20                 if H is empty then
21                     exit the inner 'while' loop;
22                 compute the overlap of the executed path of h' against the execution path
                   of each of the remaining hypotheses;
23                 h' ← a hypothesis removed from H which has the longest overlapped path
                   segments;
24                 the robot retraces to the nearest overlapping node v_x and then attempts to
                   traverse the non-overlapped part v_x,...,v_0 of h';
25     if a hypothesis is confirmed then
26         do 'loop augmentation' on S;
27     else // no hypothesis exists, or, all hypotheses are rejected
28         do 'non-loop augmentation' on S and U;
29 return S;
```

overlap of execution paths) that if a hypothesis h'' has an execution path that overlaps with the executed path of a previously examined hypothesis h' from the beginning, then the expected perception for h'' on the overlapped part $\mathcal{P}_{h''}^{E-overlap}$ can be compared against the observed perception $\mathcal{P}_{h'}^{overlap}$ on the overlapped part, and h'' is rejected if $\mathcal{P}_{h''}^{E-overlap}$ and $\mathcal{P}_{h'}^{overlap}$ do not match. To illustrate this, suppose that the executed motion for h' is (3,2,1) and h'' has motion (3,2,4,5). In this enhancement, by assuming h'' is true, the executed motions 3 and 2 of h' must map onto known vertices in the map S, because 3 and 2 visit places on the computed path of e'', which is on the map S. The observed perceptions at these overlapping known vertices $\mathcal{P}_{h'}^{overlap}$ are compared against the the signatures of the overlapped known nodes (which are the expected perceptions $\mathcal{P}_{h''}^{E-overlap}$ at these places), as in stage-1 of enhancement-1.

In order to evaluate the performance of this enhancement, experiments were conducted on the same set of lattice hole graphs and densely connected hole graphs that were used in evaluating the original algorithm and enhancement-1. Both the original algorithm, enhancement-1 (Capturing overlap of execution paths) and enhancement-2 (Exploiting executed paths that map onto S) algorithm were tested on the graphs. For enhancement-2, the algorithm sketched in Algorithm 4.2 is used, which captures and exploits the path overlap in the motion-based validation. The costs are reported in Figure 4.4, where theoretical lower bounds are also plotted. Results show that for both graphs, enhancement-2 provides further cost reductions over enhancement-1. For example, for the 90 node lattice hole graphs, enhancement-2 reduces the average cost from 13,800 steps in the original algorithm to 3,400 steps – about a 75% reduction. For the 260 node lattice hole graphs enhancement-2 reduces the average cost from 152,000 steps in the original algorithm to



(a) Lattice graphs with 10% edge removed



(b) Densely connected graphs with 10% edges removed

Figure 4.4: Performance of enhancement-2 (Exploiting executed paths that map onto S) on different graphs (log scale). Results are averaged over 30 graphs, each with randomly removed edges. Error bars show standard deviations.

24,000 steps – about a 84% reduction. Note that this enhancement requires no extra motion by the robot but does require additional computation.

Enhancement-3: Exploiting executed paths that map onto S and unknown places

In enhancement-2 (Exploiting executed paths that map on to S), when validating a hypothesis we first assume the hypothesis is true and then trace all previously executed traversals looking for inconsistencies. When a motion traverses an unexplored edge that leads to an unknown place (motion 1 in the earlier example), we stop tracing because we don't have signature information for unknown places. All of the observed perceptions thereafter ([3,A,A] and [5,A,A] in earlier example) cannot be used for comparison and thus are discarded in the current comparison. Can we also exploit the perceptions obtained at these unknown places? This is explored here. In order to exploit comparisons of perceptions made at unknown places, when the newly explored edge

e and the unknown end vertex v_u are finally validated (either as a new place or a known place) and thus the map is augmented (either by non-loop augmentation or by loop augmentation), we trace all of the previous traversals executed in validating e and v_u, starting from the validated robot place v_u and back-link (entry edge) e at v_u. For a motion that visited an unexplored edge of a known vertex, we record the corresponding perception at the unknown end as the signature of the unknown end. For the example given above, suppose after all the validations, v_u turns out to be a new vertex denoted vertex v_x of degree 6 and e is labelled as the 0'th edge of v_x. Then from the previously executed motion sequence (3,2,1,2) and its corresponding observed perception ([4,A,A], [2,A,A], [3,A,A], [5,A,A]), we can infer that the robot has traversed the previously unexplored edge 3 of the new vertex v_x. So we can record that edge 3 of vertex v_x leads to an unknown 'neighbor' with signature [4,A,A], denoting as v_x–3–[4,A,A]. Then during validation in later exploration steps, for a hypothesis $h'' = (e'', v_{k''})$ of a newly explored edge e and unknown end v_u, as in enhancement-2, we assume that h'' is true and then starting from hypothesized $v_{k''}$ and e'', trace previous motions executed in validating this newly explored edge e. Now suppose that a previously traversed motion for a hypothesis of e is (5,3,2,1), and that motion 5 (from $v_{k''}$) maps onto v_x via edge 0 and thus next motion 3 maps onto the 3rd relative edge (edge 3) of v_x. Now we can retrieve the correct signature [4,A,A] for the other end of edge 3 in v_x, even if edge 3 of vertex v_x is (still) an unexplored edge. (Edge 3 of v_x may have already become an explored edge. In such case since the end vertex is known, the signature of the other end can be retrieved from map S directly.) If the observed perception for motion (5,3,2,1) is not ([6,A,A], [4,A,A], ...), then h'' is incorrect and thus can be rejected immediately. Note that in

enhancement-2, only [6,A,A] – the signature of v_x – is used for comparison (v_x is a known place). With extra effort, now we have the additional information [4,A,A] to examine a hypothesis before executing its motion sequence. Potentially this further reduces exploration cost.

As described above we record the observed perception (signature) of unknown vertices that are adjacent to known vertices. We can extend this idea, at the cost of increased memory and computation, by recording more or all the observed perceptions (signatures) of unknown places. Here we consider recording all of the observed perceptions (signatures) of unknown places. That is, in tracing previous traversals, in addition to unknown places that are adjacent to known vertices, we also record the perception information of the unknown places that are adjacent to other unknown places. Without knowing how the unknown places connect, we index these signatures using the relative edge orderings from a known place. Consider again the above example. For the previously executed motion sequence (3,2,1,2) whose observed perception is ([4,A,A], [2,A,A], [3,A,A], [5,A,A]), as above, we record that edge 3 of v_x leads to an unknown neighbor with signature [4,A,A]. Further, we record that motion along (relative) edge 2 of this unknown neighbor leads to another unknown place with signature [2,A,A], and then relative edge 1 leads to another unknown vertex of signature [3,A,A], and then relative edge 2 leads to an unknown vertex of signature [5,A,A] (Figure 4.5). We denote this as v_x–3-[4,A,A]–2-[2,A,A]–1-[3,A,A]–2-[5,A,A]. We maintain a list of such records. Then in validating the hypothesis $h'' = (e'', v_{k''})$ of e, we assume h'' is true and trace previous motions (5,3,2,1) in validating e. Suppose as before that motion 5 (from $v_{k''}$) maps onto v_x and next motion 3 maps onto edge 3 of v_x. Now in addition to retrieving [4,A,A] for edge 3 of v_x, we can further retrieve from the maintained list of records that for next

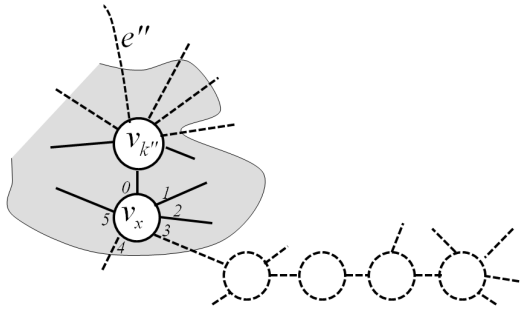


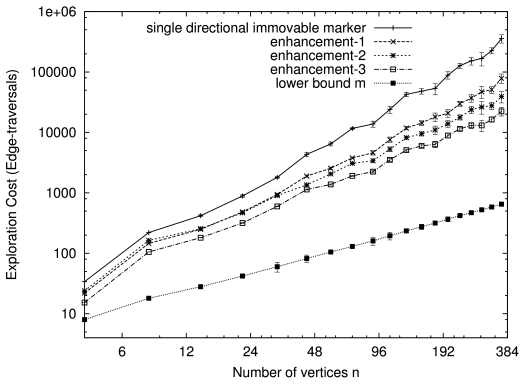
Figure 4.5: Record v_x–3-[4,A,A]–2-[2,A,A]–1-[3,A,A]–2-[5,A,A]. Assuming that hypothesis $h'' = (e'', v_{k''})$ is true, motion 5 (from $v_{k''}$) in previous motions (5,3,2,1) maps onto v_x via edge 0.

motion 2 and 1, the corresponding signatures are [2,A,A] and [3,A,A], even if the edges are (still) unexplored edges. The retrieved signatures are used for comparison with the corresponding perceptions. If the observed perception sequence is not ([6,A,A],[4,A,A], [2,A,A],[3,A,A],...) then the hypothesis h'' is rejected immediately, otherwise we validate the hypothesis with motion as in the original algorithm, or alternatively, we adopt the greedy technique used in the stage-2 of enhancement-1. This version of the enhancement is sketched in Algorithm 4.3.

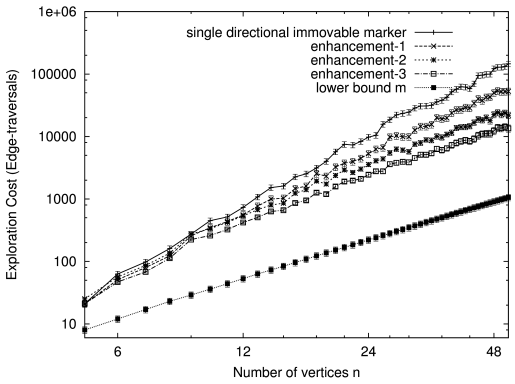
Experiments were conducted on the same sets of lattice hole graphs and densely connected hole graphs that were used in evaluating the original algorithm and the previous enhancements. Both the original algorithm, the previous two enhancements and this enhancement were run on the graphs. For this enhancement, the algorithm sketched in Algorithm 4.1 was used, which captures and exploits the overlap in the motion-based validation processes. The costs are reported in Figure 4.6. Results show that for both graphs, this enhancement provides further cost reductions over the previous two enhancements. For example, for the 90 node lattice hole graphs, this enhancement reduces the average cost from 13,800 steps in the original algorithm to 2,650 steps – about a 80% reduction. For the 260 node lattice hole graphs this enhancement reduced

Algorithm 4.3: Enhancement-3: Exploiting traversed paths that map onto S and unknown places. Stage-1 of enhancement-1 is used (line 27-29).

```
1  the robot drops the marker at v_0, pointing to an edge; S ← {v_0}; U ← edges in v_0;
2  T ← {}; // a container to store perceptions of unexplored edges
3  while U is not empty do
4      remove an unexplored edge e = (v_k, v_u) from U;
5      the robot traverses S to v_k and follows e to v_u;
6      H ← set of loop closing hypotheses h' = (e', v_k') where e' = (v_k', v_u') and the known
       end v_k' has the same signature as v_u;
7      compute motions and expected perceptions for all the hypotheses in H;
8      h' = (e', v_k') ← a hypothesis removed from H;
9      the robot attempts to traverse the computed path v_k', ..., v_0;
10     while True do
11         based on observed perception P_h' along path v.s. the expected perception P^E_h' do
12             case P_h' and P^E_h' match exactly
13                 confirm hypothesis h', exit inner 'while' loop;
14             case P_h' and P^E_h' do not match
15                 the robot stops and remains at the location;
16                 for each remaining hypothesis h'' in H do
17                     assume h'' is true, trace all the previously executed motions for e;
18                     if a motion mapped onto a known place in S then
19                         retrieve signature of the known place from S;
20                     else // mapped onto unknown place
21                         try to retrieve signature by searching T;
22                     if a signature is retrievable but does not match the corresponding
                       motion's observed perception then
23                         remove h'' from H; // reject h'' without motion!
24                 if H is empty then
25                     exit the inner 'while' loop
26                 compute the overlap of the executed path of h' against the execution path
                   of each of the remaining hypotheses;
27                 h' ← a hypothesis from H which has the longest overlapped path segments;
28                 the robot retraces to the nearest overlapping node v_x and then attempts to
                   traverse the non-overlapped part v_x,...,v_0 of h';
29     if a hypothesis is confirmed then
30         do 'loop augmentation' on S;
31     else // no hypothesis exists, or, all hypotheses are rejected
32         do 'non-loop augmentation' on S and U;
33     based on validated place v_u and back-link e, trace all executed motions in validating e,
       and add to T the perceptions from unknown places;
34 return S;
```



(a) Lattices with 10% edges removed



(b) Densely connected graphs with 10% edges removed

Figure 4.6: Performance of enhancement-3 (Exploiting traversed paths that map onto S and unknown places) on different graphs (log scale). Results are averaged over 30 graphs, each with randomly removed edges. Error bars show standard deviations.

the average cost from 152,000 steps to 15,000 steps – about a 90% reduction. The fraction of reductions of both the three enhancements on different sized graphs are summarized in Table 4.1 and Table 4.2. Note that as before, this enhancement requires no extra motion by the robot but does require additional computation. In the algorithm used for the experiments, the records of perceptions are stored in a simple linked-list like data structure and a simple sequential search is used to retrieve records. For large environment, indexing the perceived information is an important component of efficiency. Thus it is an interesting future work to develop more efficient strategies for indexing the perceived information.

4.2 Expanding local signatures

The three approaches introduced above exploit traversals of validated hypotheses. These techniques require no extra motion by the robot. So if the computation cost and memory usage

Enhancements	Size of graphs								
	72	90	108	152	202	230	260	324	360
enh-1	67.5%	67.8%	68.4%	70.1%	74.5%	75.5%	75.7%	77.4%	78.0%
enh-2	73.5%	75.3%	78.0%	80.2%	84.3%	86.0%	86.4%	87.8%	89.0%
enh-3	81.7%	82.8%	83.6%	86.0%	88.8%	89.9%	90.4%	91.9%	93.0%

Table 4.1: Average percent cost reductions of the three enhancements on 'lattice hole' graphs of different sizes.

Enhancements	Size of graphs								
	24	28	32	36	40	44	48	52	55
enh-1	45.7%	53.7%	53.8%	53.8%	57.0%	57.4%	58.9%	63.7%	63.4%
enh-2	69.7%	76.5%	78.5%	78.7%	81.2%	83.3%	84.7%	87.5%	86.5%
enh-3	75.0%	82.9%	83.3%	84.1%	86.6%	89.0%	89.9%	90.7%	91.2%

Table 4.2: Average percent cost reductions of the three enhancements on 'complete hole' graphs of different sizes.

are not the main concern, these techniques can be incorporated into the original algorithm with no additional exploration cost. These approaches attempt to reduce the cost by reducing the number of hypotheses that require motion, and otherwise by reducing the motion required in validating hypotheses. For some environments, up to 90% cost reductions are observed. Here we present a second class of approaches, which try to reduce the validation cost by expanding the local signatures. With expanded local signatures, these approaches try to improve on the original algorithm by reducing the number of potential hypotheses. These approaches require extra motion by the robot to construct an expanded local signature of a vertex and thus cannot guarantee reduced cost for all environments. That being said, experimental validation shows that for many environments these enhancements provide significant cost reduction. There are many ways to expand the local signature. We present two techniques here, each with different requirements for extra traversals.

Enhancement-4: Expanding local signatures of vertices in S

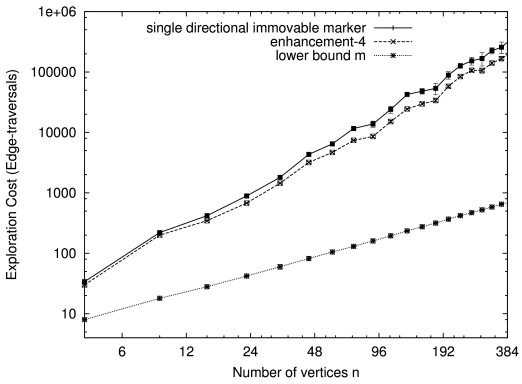
In the original algorithm, for a newly explored edge e and its unknown end v_u, each unexplored edge $e' = (v_{k'}, v_{u'})$ along with its known end $v_{k'}$ is considered a loop closing hypothesis if its known end $v_{k'}$, which is hypothesized to correspond to v_u, has the same signature as v_u. Since we lack signature information of an unexplored place such as $v_{u'}$, we ignored the fact that for the hypothesis $h' = (e', v_{k'})$ to be true, the unknown end vertex $v_{u'}$ of e' should correspond to the known end v_k of e and thus the signature of $v_{u'}$ should match that of v_k. This enhancement uses extra traversals to expand local signautures of known vertices in S to include the signatures of the unknown neighbors of each (unmarked) known vertex in S. Through the use of an expanded signature, the algorithm has the potential to validate fewer hypotheses for each newly explored edge e. In order to maintain the signature information of the neighbors of each known vertex in S, the original algorithm is modified as follows. After a newly explored vertex v_u is validated, if v_u is validated to be distinct to vertices in S and thus is added as a *new* vertex, then the robot further traverses each incident edge of v_u (except the newly explored entry edge e) to sense the signatures of the neighbors. The expanded signature information of each visited place is recorded and is used in determining potential hypotheses of the newly explored e and v_u. Then during later exploration steps, in determining if an unexplored edge $e' = (v_{k'}, v_{u'})$ in U along with $v_{k'}$ is a loop closing hypothesis, based on the maintained expanded signature of $v_{k'}$ which contains the signature of all its neighbors including $v_{u'}$, we retrieve the signature information of $v_{u'}$ and then compare the signature of $v_{u'}$ against the known signature of v_k. An unexplored edge $e' = (v_{k'}, v_{u'})$ along with its known end $v_{k'}$ is considered a potential loop closing hypothesis of e

Algorithm 4.4: Enhancement-4: Expanding local signatures of vertices in S.

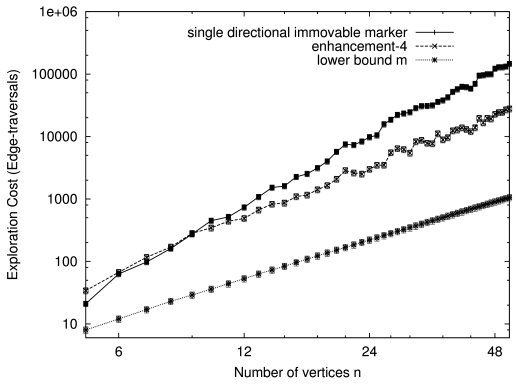
```
1  the robot drops the marker at v_0, pointing to an edge;
2  S ← {v_0}; U ← edges in v_0; // initial S & U
3  D ← {} ; // container for neighbor signature info of known places
4  while U is not empty do
5      remove an unexplored edge e = (v_k, v_u) from U;
6      the robot traverses S to v_k and follows e to v_u;
7      H ← set of loop closing hypotheses h' = (e', v_k') where e' = (v_k', v_u'), the known end
       v_k' has the same signature as v_u, and the unknown end v_u' has the same signature
       (retrieved from D) as v_k;
8      while H is not empty do
9          h' = (e', v_k') ← a hypothesis removed from H;
10         compute motions for the path v_k', ..., v_0, and P^E_h';
11         the robot attempts to traverse the path v_k', ..., v_0;
12         based on observed perception P_h' during traversal do
13             case P_h' and P^E_h' match exactly
14                 confirm h', exit inner 'while' loop;
15             case P_h' and P^E_h' do not match
16                 reject the hypothesis;
17                 the robot retraces to v_u;
18     if a hypothesis is confirmed then
19         do 'loop augmentation' on S;
20     else // no hypothesis exists, or, all hypotheses are rejected.  v_u is new.
21         do 'non-loop augmentation' on S and U;
22         the robot explores all the other neighbors of v_u and add the sensed signature info
           into D;
23 return S;
```

and v_u only if its known end $v_{k'}$ has the same signature as v_u (as in the original algorithm) *and* its unknown end $v_{u'}$ has the same signature as v_k. Otherwise e' along with $v_{k'}$ is not a potential hypothesis of e and v_u. This enhancement is sketched in Algorithm 4.4.

This enhancement requires extra traversals of incident edges at each new vertex, excluding the entry edge to the vertex. A total of $2\sum_{k=1}^{n-1}(d_k - 1) \approx 4m - 2n$ extra traversals are required, where d_k is the degree of vertex k. Clearly for fully homogeneous environments where all the



(a) Lattices with 10% edges removed



(b) Densely connected graphs with 10% edges removed

Figure 4.7: Performance of enhancement-4 (Expanding local signatures of vertices in S) on different graphs (log scale). Results are averaged over 30 graphs, each with randomly removed edges. Error bars show standard deviations.

vertices have the same degree, this enhancement provides no useful additional information and thus this enhancement does not provide a decrease in cost. Actually, it increases the cost due to the extra traversals. How does the enhancement behave in heterogeneous environments? The enhancement was run on the same sets of graphs used earlier. For lattice hole graphs where $m \leq 2n$ and thus $O(n)$ extra traversals are required, this enhancement provides some cost reductions (up to a 36% cost reduction for the tested graphs), as shown in Figure 4.7(a) and Table 4.3. It is interesting and perhaps even a little surprising to observe that this enhancement works even better on densely connected graphs. Here substantial cost reductions (more than 70%) are obtained, and as the graph size increases this reduction increases (see Figure 4.7(b) and Table 4.4). Further investigation of the results shows that although $O(n^2)$ extra traversals are required and the validation paths (to v_0) are short, substantive cost reduction results from the fact that a large number of hypotheses exist in the original algorithm, and a large fraction

Enhancements	Size of graphs								
	72	90	108	152	202	230	260	324	360
enh-4	36.5%	37.7%	36.8%	38.0%	34.8%	33.8%	30.2%	37.7%	35.3%

Table 4.3: Average percent cost reductions of the enhancements-4 on 'lattice hole' graphs of different sizes.

Enhancements	Size of graphs								
	24	28	32	36	40	44	48	52	55
enh-4	72.3%	73.7%	74.2%	78.7%	79.9%	81.2%	83.2%	86.1%	84.6%

Table 4.4: Average percent cost reductions of the enhancement-4 on 'complete hole' graphs of different sizes.

of such hypotheses are not considered as hypotheses in this enhancement, as the degree of the vertices are largely distinct, resulting in relatively unique expanded signatures.

Enhancement-5: Expanding local signatures of vertices in S and each current place

Can we reduce the number of hypotheses further? Enhancement-5 'invests' in additional traversals in an effort to further limit the potential hypotheses for a newly explored edge. As with enhancement-4 (Expanding local signatures of vertices in S), this enhancement also maintains the signature information of neighbors of each known vertex in S. In addition, this enhancement expands the signature of unknown end v_u of each newly explored edge e. In determining whether an unexplored edge $e' = (v_{k'}, v_{u'})$ along with the known end $v_{k'}$ is a potential hypothesis of e and v_u, the expanded signature of v_u is compared against the maintained expanded signature of $v_{k'}$. The original algorithm is modified as follows. After entering an unknown end vertex v_u of a newly explored edge e, the robot conducts extra traversals on each other incident edge of v_u. These extra traversals generate the neighbor signature information of v_u, enumerated from the entry

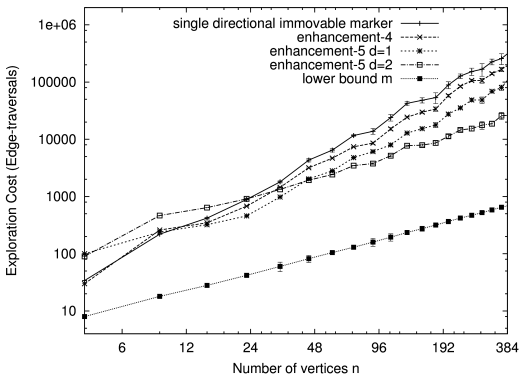
edge. For each unexplored edge e', the algorithm retrieves the neighbor signature information of $v_{k'}$, enumerated according to the enumeration rule and starting from edge e' (which should correspond to the entry edge if $(e', v_{k'})$ is the true hypothesis), and compares the expanded signature of $v_{k'}$ against the expanded signature of v_u. Clearly, for e' along with $v_{k'}$ to be a potential loop closing hypothesis of e and v_u, the expanded signature of $v_{k'}$ and that of v_u should match exactly. If not, then $(e', v_{k'})$ is not a potential loop closing hypothesis of e and v_u. For example, suppose that the robot enters a vertex v_u from a known vertex v_k which has signature [7,A,A]. At v_u the robot traverses the other edges of v_u according to the enumeration rule and senses [3,A,A],[5,A,A] and [6,A,A]. Then for each unexplored edge $e' = (v_{k'}, v_{u'})$ along with $v_{k'}$ to be a valid loop closing hypothesis, the expanded signatures of vertex $v_{k'}$, enumerated from edge e' should be exactly ([7,A,A], [3,A,A], [5,A,A], [6,A,A]). If it is not, then $(e', v_{k'})$ is not a valid hypothesis. In order to maintain the neighbor signature information of each known vertex, as in enhancement-4 (Expanding local signatures of vertices in S), we record the signature if v_u is validated to be a new place. Unlike enhancement-4, since the robot has already traversed the neighbors of v_u when it first entered v_u, we just record the signatures of v_u's neighbors as the expanded signature of this newly explored vertex, which can be used for later validations. No additional motion is required. Note that in comparing the neighbor degree information, this enhancement incorporates the idea of the enhancement-4 (Expanding local signatures of vertices in S) that for an unexplored edge $e' = (v_{k'}, v_{u'})$ along with $v_{k'}$ to be a hypothesis, the unknown end $v_{u'}$ must have the same (local) signature as v_k. Above we expanded the local signature to the immediate neighbors (call this a $d = 1$ signature). We can expand the signature of a vertex to

include the neighbors of its neighbors (call this a $d = 2$ signature). More generally, by investing in more traversals we can expand the signature of a vertex to include the signatures of neighbors of distance $d \geq 2$. This enhancement is sketched in Algorithm 4.5.

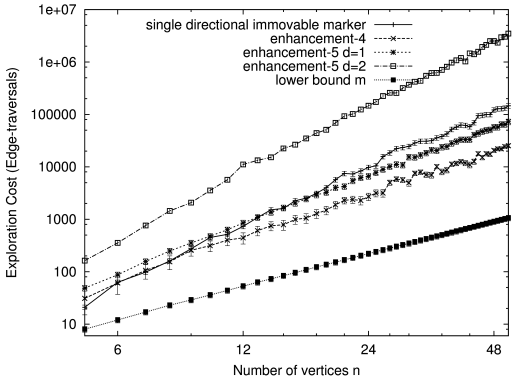
Algorithm 4.5: Enhancement-5: Expanding local signatures of vertices in S and each current place.

```
1  the robot drops the marker at v_0, pointing to an edge;
2  S ← {v_0}; U ← edges in v_0; // initial S & U
3  D ← {} ; // container for neighbor degree info of known places
4  while U is not empty do
5      remove an unexplored edge e = (v_k, v_u) from U;
6      the robot traverses S to v_k and follows e to v_u;
7      the robot explores all the distance d neighbors of v_u to obtain the neighbor degree info
       (expanded local signature of v_u);
8      H ← set of loop closing hypotheses h' = (e', v_k') where e' = (v_k', v_u') and known end
       v_k' has the same expanded signature (retrieved from D) as that of v_u (just explored);
9      while H is not empty do
10         h' = (e', v_k') ← a hypothesis removed from H;
11         compute motions for the path v_k', ..., v_0, and P^E_h';
12         the robot attempts to traverse the path v_k', ..., v_0;
13         based on observed perception P_h' during traversal do
14             case P_h' and P^E_h' match exactly
15                 confirm e', exit inner 'while' loop;
16             case P_h' and P^E_h' do not match
17                 the robot retraces to v_u;
18                 reject the hypothesis and continue;
19     if a hypothesis is confirmed then
20         do 'loop augmentation' on S;
21     else // no hypothesis exists, or, all hypotheses are rejected
22         do 'non-loop augmentation' on S and U;
23         add the expanded signature info of v_u into D; // already explored.  just record.
24 return S;
```

This enhancement requires extra traversals for each explored edge. In contrast to enhancement-4 (Expanding local signatures of vertices in S) where extra traversals of neighbors are conducted



(a) Lattice graphs with 10% edges removed



(b) Densely connected graphs with 10% edges removed

Figure 4.8: Performance of enhancement-5 (Expanding local signatures of vertices in S and each current place) on different graphs (log scale). Results are averaged over 30 graphs, each with randomly removed edges. Error bars show standard deviations.

at each new vertex v_u once, here extra traversals may be conducted at each newly explored v_u multiple times (every time the vertex is explored). So for the distance $d = 1$ version, $O(md_{max})$ extra traversals may be required, where d_{max} is the maximum degree in the graph. In homogeneous environments this enhancement results in increased cost (even higher than for enhancement-4). For heterogeneous graphs, this enhancement is evaluated on the same set of graphs used earlier, using both the $d = 1$ version and $d = 2$ version of the enhancement. Results on lattice hole graphs are shown in Figure 4.8(a) and Table 4.5. We can see that the $d = 1$ version of the enhancement, which has $O(n)$ extra traversals, provides further cost reduction than enhancement-4. The $d = 2$ version of the enhancement which may require additional 'investment', results in a higher cost when the graph size is small. When the graph size is sufficiently large (more than 50 vertices in the experiments reported here) the $d = 2$ version enhancement results in a lower cost than the $d = 1$ version enhancement, and the reduction increases as the graph size increases. In the

Enhancements	Size of graphs								
	72	90	108	152	202	230	260	324	360
enh-4	36.5%	37.7%	36.8%	38.0%	34.8%	33.8%	30.2%	37.7%	35.3%
enh-5 d=1	58.9%	55.7%	66.7%	68.1%	69.0%	72.1%	68.2%	69.5%	68.9%
enh-5 d=2	68.6%	71.2%	77.3%	82.6%	86.6%	87.9%	87.8%	89.4%	91.2%

Table 4.5: Average cost reductions of the three enhancements on 'lattice hole' graphs of different sizes.

Enhancements	Size of graphs							
	28	32	36	40	44	48	52	55
enh-4	73.7%	74.2%	78.7%	79.9%	81.2%	83.2%	86.1%	84.6%
enh-5 d=1	37.3%	31.9%	21.5%	34.1%	42.0%	41.5%	37.9%	39.7%
enh-5 d=2	-1145%	-1454%	-1944%	-1805%	-1970%	-1962%	-2291%	-2372%

Table 4.6: Average percent cost reductions of the enhancements on 'complete hole' graphs of different sizes.

experiments up to a 90% cost reduction is observed. On densely connected graphs, the $d = 1$ signature version of enhancement-5, which may require $O(n^3)$ extra traversals on these graphs, gives a cost reduction after a certain graph size, but the reduction is smaller than that generated by enhancement-4. The $d = 2$ signature version of this enhancement, which requires $O(n^4)$ extra traversals on these graphs, results in a much higher cost than the original algorithm! The results are shown in Figure 4.8(b) and Table 4.6. The results show that enhancement-5 (especially the $d = 2$ version), does not provide an improvement on such densely connected graphs. This result is not surprising since the cost reduction from the validate steps is countered by the large number of extra traversals required in order to construct the expanded signatures.

4.3 Combining the two classes of enhancements

The above sections present two different classes of enhancements. The first class of enhancements tries to reduce the number of hypothesis that need to be validated, and the number of motions that must be executed. Enhancement-3 (Exploiting executed paths that map onto S and unknown places) is the 'best' for both lattice graphs and densely-connected graphs, producing up to a 90% cost reduction on both type of graphs. The second class of enhancements tries to reduce the number of potential hypothesis. Enhancement-5 (Expanding local signatures of vertices in S and each current place) with $d = 2$ expanded signature gives the best performance on lattice graphs, producing up to a 90% reduction, whereas enhancement-4 (Expanding local signatures of vertices in S) works well for densely connect graph, producing up to a 85% cost reduction.

It is interesting to consider combining these two classes of approaches in the hope that the overall cost can be further reduced. The 'best' algorithms from the two classes of enhancements are selected, and are combined in a strategic manner. In particular, for lattice hole graphs enhancement-3 (Exploiting executed paths that map onto S and unknown places, with stage-2 of enhancement-1 incorporated) and $d = 2$ version of enhancement-5 (Expanding local signatures of vertices in S and each current place) are selected and combined – denote it enhancement 3+5. For densely connected graphs, the best enhancements, namely enhancement-3 (Exploiting executed paths that map onto S and unknown places) and enhancement-4 (Expanding local signatures of vertices in S), are combined –denote it enhancement 3+4. Results are shown in Table 4.7 and 4.8, which demonstrate that for both environments the two classes of enhancements can be combined with some enhanced performance.

Enhancements	Size of graphs								
	72	90	108	152	202	230	260	324	360
enh-3	81.7%	82.8%	83.6%	86.0%	88.8%	89.9%	90.4%	91.9%	93.0%
enh-5 d=2	68.6%	71.2%	77.3%	82.6%	86.6%	87.9%	87.8%	89.4%	91.2%
enh 3+5	**83.1%**	**83.1%**	**85.1%**	**87.6%**	**90.1%**	**90.8%**	**91.9%**	**93.0%**	**94.0%**

Table 4.7: Average cost reductions of enhancement 3+5 on lattice hole graphs of different sizes.

Enhancements	Size of graphs								
	24	28	32	36	40	44	48	52	55
enh-3	75.0%	82.9%	83.3%	84.1%	86.6%	89.0%	89.9%	90.7%	91.2%
enh-4	72.3%	73.7%	74.2%	78.7%	79.9%	81.2%	83.2%	86.1%	84.6%
enh 3+4	**78.9%**	**86.7%**	**86.7%**	**86.8%**	**89.3%**	**91.4%**	**91.9%**	**92.1%**	**92.8%**

Table 4.8: Average cost reductions of enhancements 3+4 on complete hole graphs.

4.4 Summary

This chapter presented two classes of enhancements to the basic directional lighthouse algorithm described in the previous chapter. The first class exploits executed traversals to reject potential hypotheses. These approaches try to reduce the number of hypotheses that require motion, and the amount of motion required in validating the hypotheses. These approaches require no extra edge traversals by the robot. The second class of approaches requires extra traversals to construct expanded local signatures. With the expanded local signatures, the number of hypotheses of a newly explored edge can be potentially reduced. In the second class of enhancements, we consider signatures based on physically exploring neighborhoods of different ranges. This can provide enhanced performance, but at the risk of executing additional motions of the robot. Finally, this chapter looks at the potential of integrating these two approaches. The integration of the various techniques can provide additional improvement over each class of enhancement separately, although there is a smaller return to this additional complexity. In the second class

of enhancements, we consider expanded signatures up to a distance 2. For some environments, signatures of distance $d = 2$ provide further cost reduction over signatures of $d = 1$. Generally, signatures of larger distance d provide more information to distinguish places but the potential cost reduction may be countered by the larger number of extra traversals required in order to construct the expanded signatures. Moreover, the number of traversals required to construct the signatures has the potential to explode. An interesting direction for future work is to explore the boundary of expanding signatures for different environments, as well as the scalability of expanding the signatures for different environments. Another interesting direction for future work is to characterize graph properties that influence the performance of each of the enhancements described in this chapter.

Chapter 5

Mapping with less marker information

It was shown in Chapter 3 that a single directional immovable vertex marker is sufficient to map the world deterministically. A single directional immovable marker establishes both explicit place information and back-link information at the marked vertex. In this chapter we consider mapping with less information. This chapter first considers mapping with a single undirected immovable marker, and then investigates mapping with no markers. In both cases the hypothesis-based approach described in Chapter 3 is adapted to deal with insufficient marker information.

5.1 Mapping with a single undirected immovable marker

A key observation of Chapter 3 is that if the world contains some structure that forms a unique signature which provides both explicit place and back-link information, then the world can be mapped deterministically. Since a single undirected immovable marker can be used to provide explicit place information at a vertex, we are assured that as long as there exists a vertex in the world that can provide explicit back-link information, then this vertex can be used (marked) to establish a directional lighthouse, enabling the robot to map the world deterministically. There are many situations where explict back-link information exists at a vertex. Here we present

(a) (b)

Figure 5.1: Graphs with some structures can be mapped with an undirected immovable marker. (a) Any vertex with degree one. (b) Any vertex with a neighbor whose degree differs from other neighbors. The marked vertex has neighbors with degree 2, 3, 4 and 4. The neighbor with degree 2 or 3 can be used to establish explicit back-link information at the vertex.

one such class. We have seen earlier that a vertex of degree one provides explicit back-link information at the vertex (Figure 5.1(a)). Once marked as a unique location by the marker, the vertex can serve as a directional lighthouse. Such a vertex is sufficient but not necessary. By expanding the local signatures, we can generalize this type of structure. For example, by expanding the signature of a vertex to include the signatures of its adjacent vertices (i.e., expand to distance $d = 1$ neighbors), explicit back-link information exists in a vertex if among the signatures (degrees) of all the neighbors, at least one signature is unique, i.e., the vertex has at least one neighbor vertex whose degree is different from that of all the other neighbors. For such a vertex, a neighbor having a unique degree establishes at this vertex an absolute ordering information from which explicit back-link information can be retrieved. An example is shown in Figure 5.1(b). More generally, suppose on encountering a vertex the robot visits each of the adjacent vertices according to the enumeration rule, constructing the sequence $D = (d_1, ..., d_k)$ of the degrees of the neighboring vertices of the vertex. For this vertex to serve as a lighthouse vertex it must be unique (we can satisfy this requirement using the undirected marker), and it must also provide explicit back-link information. This vertex will provide explicit back-link

information if it is not the case that there exists a cyclic rotation of D in the embedding that maps onto D, except for a complete rotation. Note that the 'single leaf' vertex and the vertex with a unique degree d_i in its D meet this test, but many other local embeddings meet this requirement as well. More situations exist if we further extend the local signature of a vertex by considering larger neighborhood of the vertex.

Based on the above observation, mapping with an undirected vertex marker can be conducted by having the robot first conduct a random walk searching for an appropriate vertex. Upon entering a vertex, the robot traverses each other incident edge at the vertex based on the enumerating rule, retrieving the sequence D of the degree information of the neighbors. The robot then examines the sequence D to see if the sequence of degrees meets the requirement in terms of rotational equivalence. If this vertex lacks rotational equivalent then it can serve as a directional lighthouse. The robot stops the random walk, dropping the marker at this vertex and then explores the world using the deterministic single directional marker algorithm or one of its enhancements. If this vertex does not meet the test then the robot continues the random walk until either a vertex that meets the test is found or otherwise some 'giving up' condition is met. In the latter case the algorithm reports failure.

Any vertex can be used as a lighthouse vertex as long as its expanded signatures contain no rotational equivalent. Experiments were conducted with $d = 1$ neighborhood expanded signatures. In the experiments, the robot starts a random walk at a randomly selected place. In examining possible direction information at a vertex, we simply considered the $d = 1$ neighborhood of the vertex. A threshold of 200 steps is set for the random walk: if the robot cannot

find a lighthouse within the threshold, the algorithm terminates reporting failure. The basic single directional immovable marker algorithm was slightly modified so that explicit back-link information can be properly retrieved and used. It becomes more expensive to validate back-link as this now requires visiting adjacent vertices to determine relative orientation to the marker. Instead of transiting every incident edge, computing the neighbor degree information and then computing the appropriate entry edge with respect to the embedding, some enhancement was developed by observing that the robot does not need to transit to all edges to complete the full vertex sequence, but only as many as necessary to determine if it entered the lighthouse with the expected entry edge. Results show that on the given lattice hole graphs and random graphs, the algorithm finds a lighthouse either at the initial location or otherwise within a few steps (usually less than 50 steps), mapping the world deterministically. It is interesting to note that this algorithm works successfully on the two different embedded graphs shown in Figure 3.4, which cannot be distinguished with a single undirected marker placed in the center vertex. When running the algorithm on these graphs, during the random walk the robot always selects one of the four corner vertices as the lighthouse, because the neighbor degrees sequence $D = (4, 3, 3)$ of the corners lacks rotational equivalent and thus contains explicit back-link information.

5.2 Mapping with no markers: probabilistic exploration

We have developed deterministic algorithms with powerful markers, and algorithms that operate with weak markers. Here we examine the case where the robot has no marker at all. Without a marker, the robot cannot map a world deterministically. Here we present some approaches to

mapping a topological world probabilistically. Without a marker, we must exploit the signatures in the environment to do validations. Here we describe two approaches of exploiting the signature information in the environment. Some of the previous techniques used in marker-based exploration are adopted here.

5.2.1 Exploiting local signatures

This approach uses local signatures or slightly expanded local signatures to select a 'likely' directional lighthouse, and then maps the world using the single directional marker algorithm or one of its enhancements. Specifically, we first have the robot conduct a random walk for a certain number of steps, and then select the traversed vertex whose local signature is believed to be (most likely) unique and that also provides back-link information. This vertex is then used as a directional lighthouse. Different selection schemes can be developed. For example, we can record the pair of degrees of adjacent vertices that the robot encounters during its random walk. After the random walk, we consider the pair of degrees that was encountered the least number of times as the basis of a potential directional lighthouse. If the least number of visits is below a threshold then this pair of degrees is selected to establish a directional lighthouse. Otherwise the algorithm terminates reporting failure. If a directional lighthouse is selected, then the robot returns to a pair of adjacent vertices that has the selected degree pair, where one of the adjacent vertices is used as the lighthouse and the other is used to establish the back-link information at the lighthouse. This approach cannot guarantee the uniqueness of the place information as well as the back-link information at the selected lighthouse. Another example selection criterion uses

expanded local signatures. At each vertex visited during random walk, the robot visits all the neighbors and establishes a list $D = (d_1, ..., d_k)$ of degrees. After the random walk, we collect the lists that contain explicit back-link information, using the rotation equivalence test given in the previous section. Then, among the collected degree lists, we treat lists that are cyclic rotations of each other as the same list and add up their visit counts (e.g., (2,3,3), (3,3,2) and (3,2,3) are considered the same list, as are (2,3,6) and (3,6,2)). If the minimal associated visit count is below a threshold, then the vertex that contains this list is selected as the lighthouse vertex. The robot then returns to a place with the selected signature (neighbor degree list), and explores the world using the single directional marker algorithm. This approach can guarantee the explicitness of back-link information at the selected lighthouse but cannot guarantee the uniqueness of the lighthouse itself. For these and other related schemes, as we cannot guarantee the explicitness of place or back-link or both, there are a number of possible outcomes of the algorithm: No potential lighthouse may be identified, and the algorithm reports failure; The selected lighthouse happens to be unique and the back-link informaiton is also explcit, and the correct world model is generated; The lighthouse (position) and/or the back-link information at it are not explicit, in which case the algorithm either detects some signature inconsistencies (due to wrong loop closing) and thus cannot proceed reporting failure, or, never terminates (when every newly visited place is always validated as a new place), or terminates with an incorrect world model. Without additional information about the world, when a world model is generated, we cannot determine whether it is a correct model or not. Note that this approach may generate a wrong world model on the two graphs shown in Figure 3.4. In both the two graphs, none of

the nodes contains both explicit place and back-link information. (Each of the four corner nodes contains explicit back-link information but ambiguous place information, whereas the center node contains explicit place information but ambiguous back-link information.)

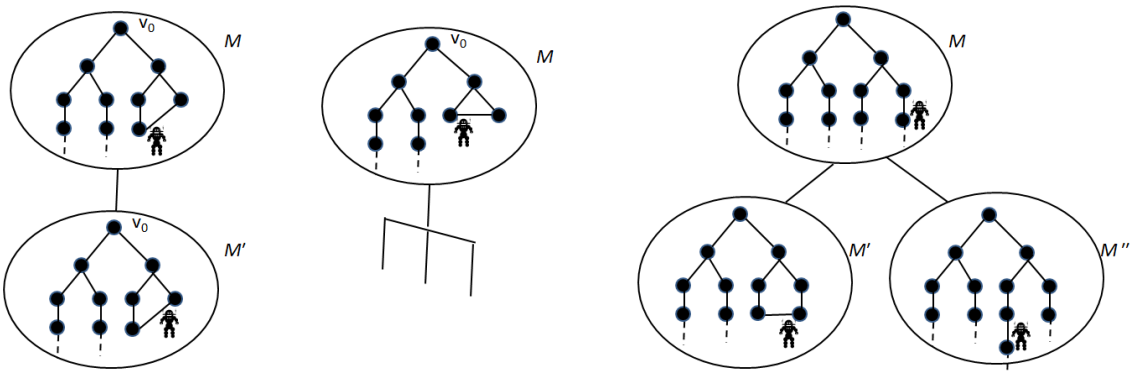
5.2.2 Exploiting global signatures

The approach described above uses a very localized signature (neighbor distance $d = 1$) to select the 'most likely' directional lighthouse. In the single directional marker algorithm used by the approach, only one world model is maintained. If the lighthouse is not unique, the algorithm may generate an incorrect world model. While this approach might be improved using signatures of neighbor distance $d > 1$, here we present an alternative approach that exploits the signature of neighbor distance $d = D$, where D denotes the diameter of the graph. In this approach signatures of a vertex is expanded to include all of the known vertices in the graph. Call this a node's 'global signature'. Similar to the ideas presented in [23], at each step we maintain multiple world models (hypotheses), which are all possible world models that are consistent with the perceptions that the robot obtained so far. One of the world models is the true (partial or complete) representation of the environment, but the algorithm does not know which one it is. After each step of exploration, we process each of the current world models generated from the previous step. The goal is to process the current world models effectively so that the true world model is maintained and updated correctly, and moreover, that incorrect models are detected and eliminated as much as possible. Several other challenging problems also have to be considered, including how the robot chooses the next edge/vertex to explore, and when the robot should

terminate the exploration.

The first challenging problem involves how to select the next 'unexplored' edge for the robot to explore. Without a true partial world model to guide the exploration, it is not trivial to explore in such a way that we maintain progress (i.e., the robot does not explore the same sub-areas forever) and moreover, guarantee completeness of the exploration (i.e., eventually all the places in the environment are explored). Here we adopt a conservative technique, which uses a breadth first search (BFS) traversal of the environments. During exploration, in addition to the maintained world models, the robot also maintains a 'traversal map' *TM*. Same as other world models, initially TM contains the starting location, but it grows with a new edge and a vertex in each step. That is, although each newly explored edge and end vertex may or may not be new places, they are always treated as new places and are added into the traversal map TM (non-loop augmentation). In each step, according to a BFS on TM, the robot traverses to a frontier vertex on TM and then explores an 'unexplored' edge of the frontier. While the BFS traversal cannot avoid duplicated explorations on the environment, it can guarantee that a progress is made sooner or later, and that eventually all of the environment is visited.

After each exploration step of the robot, we process the set of world models generated in the previous step. As in [23], the world models are maintained in a model tree T, where each model is a node in the tree. In processing these world models, we want to maintain the set of possible world models that are in agreement with the perception information, including the true world model (although we do not know which one it is). At the same time we want to eliminate inconsistent models as much as possible. Lacking any marking aids, we must exploit



(a) Situation 1. (b) Situation 2. (c) Situation 3.

Figure 5.2: Three situations on world models for new motions (0,1,1,1,1) and the obtained perceptions (2,3,2,2,2). (a) Motions (0,1,1,1,1) map onto known places of world model M, and the obtained perceptions (2,3,2,2,2) match the signatures of the known places in M. A new world model M' is duplicated from M with updated robot pose and back-link in model M'. (b) Motions (0,1,1,1,1) map onto known places of world model M but perceptions (2,3,2,2,2) do not match signatures in M. Inconsistency is detected, no new world model is generated from M. (c) Motions map onto unknown places in M. Have to do validations for loop closing hypotheses. Here $c = 1$ loop closing hypothesis is accepted. Totally $c+1$ new world models are generated, as a non-loop world model (M'' here) is always added as the (last) child model of M.

the environment structure to help process the world models. For each current world model M, knowing the previous place and back-link (entry edge) of the robot in M, we first map the new robot motions (which correspond to traversing the TM to a frontier vertex and then exploring the unexplored edge) onto the current world model M, and distinguish three possible situations: (1) the new robot motions (including the last motion of edge exploration) visit known places in the model M, and the perceptions that the robot obtained match the signatures of these places in M. (2) the new motions visit known places in the model M, but the perceptions that the robot obtained do not fully match the corresponding signatures in M. (3) the last motion traverses an unexplored edge in M. That is, the newly explored edge corresponds to an unexplored edge in M.

An example is shown in Figure 5.2 where executed motions and the obtained perceptions result in different situations in the current world models. In case (1), no inconsistency is observed. We 'pass' the model M. That is, we generate a new model M' that duplicates the current mode M, but with the updated place and back-link of the robot in the model M', and then add the new model M' into the model tree as a child model of M (Figure 5.2(a)). Then we proceed to process other current world models. In this case one new world model is generated from M. In case (2) an inconsistency is detected, M is an incorrect world model. So we void M immediately (Figure 5.2(b)). No new model is generated from M and we proceed to process other world models. In case (3) we need to conduct validations without any marker. This is the most challenging part. In the simplest approach, all the loop closing possibilities based on the local degree information are considered, and new world models are generated for each of these possibilities. Specifically, each unexplored edge in M whose known end has the same degree as the sensed degree of the robot is considered a loop closing solution, and a world model is generated for each of the loop closing possibilities. And finally, a 'non-loop' model with a new edge and a new vertex is generated. This approach guarantees that the true world model is maintained but results in the model tree growing exponentially for many environments, making the algorithm intractable. Here we try to reduce the number of loop closing possibilities using global signatures. The idea is that for each potential loop closing possibility (hypothesis) based on the local signature, instead of accepting it, we attempt to validate it by computing paths from the hypothesized location of the robot to *each* of the known places in M, and then having the robot traverse each of the paths. That is, each of the known vertices serves as a 'lighthouse'.

The robot traverses each path and compares the obtained perceptions against the expected perceptions. The expected perception at each vertex along the path contains the local signature of the vertex, and moreover, if the vertex's neighborhood contains explicit back-lin information due to a lack of rotation equivalence, also contains the expected entry edge information upon entering the vertex. For a loop closing hypothesis to be accepted, paths to *all* the 'lighthouses' must be completed with no signature mismatch. If any of the paths cannot be completed due to an observation of any mismatch, the hypothesis is rejected immediately, and the next hypothesis is examined. If all the paths are completed and no mismatch is detected, then the loop closing hypothesis is accepted. For this accepted hypothesis, a new model M' is generated with a new loop and is added into the model tree as a child of model M. Now unlike in the marker-based case where it is guaranteed that once a hypothesis is accepted then a loop is closed correctly so no other hypotheses need to be examined, here since we cannot guarantee the uniqueness of any lighthouse, the algorithm then proceeds to validate the next loop closing hypothesis (if any). Multiple hypotheses may be accepted and new models generated for each accepted hypothesis and added to the model tree. When all the hypotheses have been processed, no matter how many of them are accepted, we must also consider the possibility that the robot may have just explored a new place and an edge. Thus, a new map is generated, non-loop augmented, and added into the tree as the (last) children of M. Processing of the world model M is now completed. In this case if $c \geq 0$ loop closing hypotheses are accepted, then a total of $c + 1$ new world models are generated from M (Figure 5.2(c)). We then proceed to process other world models until all the current models have been processed. In the next step, only world models generated in

this step are processed. A simplified version of the world model processing steps is sketched in Algorithm 5.1.

The algorithm has been implemented and evaluated on a number of input graphs. In the implementation of the algorithm, several enhancement techniques developed in previous chapters were used to make the algorithm more efficient. For example, using the technique in Chapter 4, the path overlap is captured in the multiple lighthouse traversals in order to reduce the number of traversals. The implementation also considers another challenging problem: when should the robot terminate exploration? After sufficient exploration, all the places in the world would have been explored but the robot will continue to traverse in the environment in a BFS fashion. When the robot has fully explored the world, the true world model should be a completed model (i.e., no unexplored edges exist) but the other models may not be. As the robot continues in the world, the robot keeps traversing known places in the true world model with no signature mismatch (i.e., case (1), the algorithm keeps passing the true model with updated robot place and back-link on it). Based on this, the following heuristic for terminating the algorithm is followed: when a world model M becomes a completed model, and remains as the only completed model for a certain number of steps (e.g., 100 steps), the robot terminates the exploration and returns M as the likely model.

Despite the fact that global signatures are considered here, the algorithm is still not deterministic in that it may generate multiple world models and we cannot determine which world model is the correct one. Through the use of global signatures, the growth of the model tree is expected to be reduced for sufficiently heterogeneous graphs. This is confirmed on many lattice

Algorithm 5.1: Mapping with no markers, world model processing process

```
1  Based on BFS, the robot traverses the 'traversal map' TM to a known vertex v_k,
   executing motion M, and then traverses an unexplored edge e = (v_k, v_u) of v_k;
2  non-loop augmentation TM with e and v_u; // ALWAYS non-loop augmentation;
   // now process each world model generated in last move;
3  l ← level of world models generated in the previous step (in the model tree T);
4  for each world model M at level l of the model tree T do
5      traverse M 'virtually' according to the executed motion M, comparing signatures
       along the path in M against real perceptions obtained in executing M;
6      if signatures do not match along the path then
7          ; // void M, proceed to process remaining level l model in T;
       // signature match, now examine the traversed edge e;
8      if traversed edge e corresponds to an explored edge in M then
9          if other end signature on M does not match corresponding perception then
10             ; // void M, proceed to remaining level l model in T;
11         else // signature matches. 'Pass' M. A duplicated map into next level
12             new map M' = M;
13             add M' into T as a child model of M (at level l + 1);
               // now proceed to other model (if any);
14     else
           // e corresponds to an unexplored edge in M, hard work for disambiguation;
15         for each loop closing hypothesis (unexplored edge) e' = (v_k', v_u') in M where
           signature of v_k' matches that of v_u do
16             for each known vertex v_l in M other than v_k' do
17                 compute motions for a path from v_k' to v_l;
18                 the robot attempts to execute the computed motions;
19                 if signature does not match during or at the end of path execution then
20                     reject the hypothesis, exit inner for loop;
                   // else, continue executing next path;
21             if a hypothesis e' is rejected then
22                 ; // proceed to next loop closing hypothesis;
23             else // all paths completed successfully, accept this hypotheses
24                 M'= M;
25                 do 'loop augmentation' on M';
26                 add M' to T as a child of M (at level l + 1);
               // in both cases, now examine another loop closing hypothesis;
           // finally always add a new edge and vertex;
27         M'= M; 'non-loop augmentation' on M';
28         add M' to T as (last) child of M (at level l + 1);
           // now proceed to process remaining level l model;
```

hole and random graphs tested experimentally. For many graphs such as random graphs and lattices with 10% – 20% holes, the algorithm terminates according to the termination condition given above. When the algorithm terminates, the sole completed world model maintained turns out to be the correct world model. For some less heterogeneous graphs, multiple complete world models are generated but the growth of the model tree is still limited. As a final example, consider again the two embedded graphs from Figure 3.4. When exploring any one of them - e.g., the graph in Figure 3.4(a) - using only local signatures in processing multiple world models, the model tree grows quickly and among the world models more than 50 models become completed. When exploring it using global signatures, fewer models are maintained among which only six world models become completed. These complete models are shown in Figure 5.3, which include the correct model. (One of the incorrect models is the correct representation of the graph in Figure 3.4(b).) Similar results are observed when exploring the graph in Figure 3.4(b).

5.3 Summary

This chapter considers mapping topological environments where no directional lighthouse can be established. We first considered mapping with an undirected immovable vertex marker. The key observation here is that while a single undirected immovable marker is not sufficient in general, it can be used to map the world that contains some structure that, once marked, forms a unique signature which provides both explicit place and back-link information. Such a world can be mapped deterministically with an undirected vertex marker, using the single directional marker algorithm developed in Chapter 3 or one of the enhancements described in Chapter 4. Since the

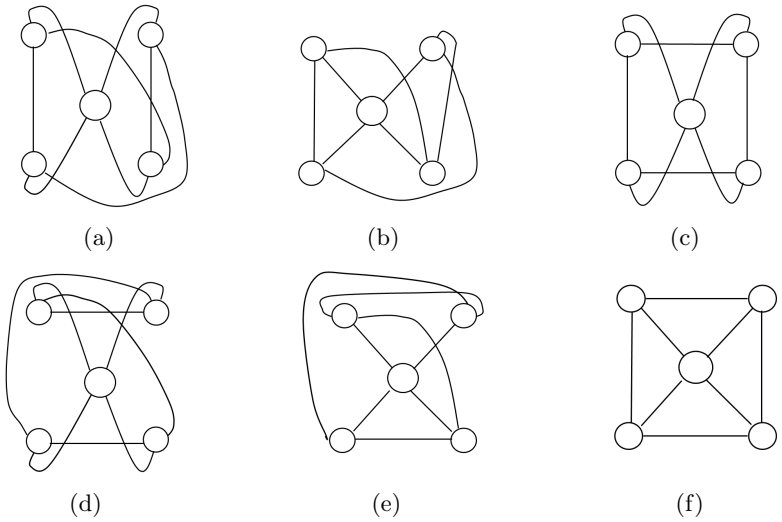


Figure 5.3: Completed world models produced by the 'global signature' algorithm on the graph in Figure 3.4(a). Listed according to the order they become completed. (f) is the correct world model. Similar results are obtained when running the algorithm on graph in Figure 3.4(b).

single undirected vertex marker can be used to provide explicit place information at a vertex, we are assured that as long as there exists a vertex in the world that can provide explicit back-link information, then this vertex can be used (marked) to establish a directional lighthouse, enabling the robot to map the world deterministically. In the experiments the robot found such a lighthouse vertex on most of the given non-homogeneous graphs. It is an interesting direction for future work to investigate whether there is a theoretical guarantee for the robot to find such a vertex in an non-homogeneous graph. Existing research on symmetry breaking of graphs (e.g., [2, 57]) might be helpful here.

This chapter also considers mapping probabilistically without any markers. Without any marker, we must rely on the structure of the environment. We examine exploiting local and global signatures during exploration. Both approaches are non-deterministic in that the algo-

rithms either cannot generate a valid world model or generate one or more world models but cannot determine which model is the true representation of the world. The challenges faced by the approaches and other marker-less approaches indicate the importance of markers or other additional information about the environment, which can be used as a cue to limit the growth of the exploration tree and to cease exploration even though some possible models have not been fully explored. Prior information is clearly useful. For example, if we know that there exists exactly one 'leaf vertex' in the world, then by searching and using the leaf vertex as the directional lighthouse the world can be mapped deterministically. Other information could be used as a cue, although such cues do not necessarily lead to a deterministic mapping of the environment. As discussed in earlier literature including [23], one cue is the number of vertices n in the environment. This information helps reject models that have more than n vertices. This information also indicates that the exploration process can terminate as soon as all the current models have n vertices. The number of edges m in the environment can be used similarly. The diameter d of the environment is another factor that can be helpful. We can reject loop closing hypotheses that result in a model with too large a diameter. Moreover, this information also helps us determine the termination condition. Since the breadth first traversal of a finite graph visits all its verteices after at most depth d, we should cease exploration after depth d of the traversal map TM has been traversed. Planarity of the environment is another cue that can help trimming the model tree. If we know that the environment is a planar graph, then we can reject the loop closing hypotheses that would generate a model that is not planar, and as in [29] we can exploit the information during exploration. Another potential class of prior knowledge is the probabilistic

distribution of some environmental properties. As mentioned in [23], the models maintained in the model tree correspond to assumptions regarding the existence (or non-existence) of multiple locations in the world that are perceptually indistinguishable. If the likelihood of such occurrences can be estimated using the probabilistic distribution information of some environmental properties, then the world models in the tree can be ranked in terms of their overall likelihood.

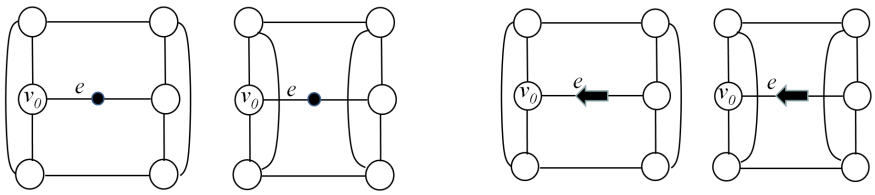
Chapter 6

Other kinds of markers

This chapter takes a divergence from the main contribution of this work and examines the relative power of markers on edges, multiple markers, and of extended markers that can be used to mark sequences of vertices and edges (string/thread markers). Portions of this chapter have appeared in the literature [63, 64, 65, 66].

6.1 Mapping with a single edge marker

The work presented elsewhere in this thesis deals with vertex-based markers. Assume instead that the robot can drop the marker along an edge. As with vertex-based markers (V-markers), edge-based markers (E-markers) can be directional or undirected. In case of a directional marker, the marker head points to one of the end vertices of the marked edge. Assume that the robot can sense the marker when it is in one of its end vertices. That is, at a vertex the robot can sense the presence or absence of the marker on the incident edges of the vertex, and can also determine the ordering of the edge upon which the marker exists (relative to the entry edge). The direction of a directional E-marker provides an orientation along the edge, e.g., the end vertex indicated by the marker head. Assume that the robot can sense this direction when it is in one of the end



(a) (b)

Figure 6.1: (a) Different embedded graphs that are not distinguishable with an undirected immovable E-marker. The robot is initially in v_0. Use the 'otherInfo' field of the general perception denotation to encode the marker information of the E-marker. Motion sequence $\mathcal{M}$=(1,1,2,2,1) results in perception $\mathcal{P}$=([3,A,A], [3,A,A], [3,A,A], [3,A,A], [3,A,E-1]) on both the two graphs, where [3, A,E-1] denotes that the E-marker is present and is located on the next edge to the left of the entry edge. (b) The graphs are distinguishable with a directional immovable E-marker. Motion sequence $\mathcal{M}$=(1,1,2,2,1) results in different perceptions ([3,A,A], [3,A,A], [3,A,A], [3,A,A], [3,A,E-1-H]) and ([3,A,A], [3,A,A], [3,A,A], [3,A,A], [3,A,E-1-T]) where H/T indicates that it is the head/tail of the marker that points toward the vertex where the robot is in.

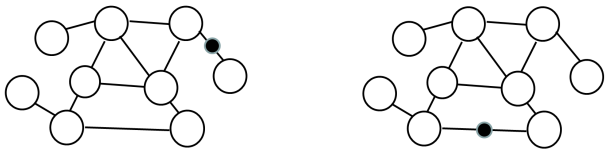
vertices, i.e., whether the marker head/tail points to the vertex the robot is in.

Similar to an undirected V-marker, an undirected E-marker is not sufficient for a robot to map an arbitrary world deterministically, although again similar to the V-marker case, a directional E-marker is. Consider the two different embedded graphs shown in Figure 6.1(a), which are not isomorphic to each other under the extended definition of graph isomorphism. Each graph has one edge that is marked with an undirected immovable E-marker. Under our perception model, the robot can sense the marked edge when it is in one of the end vertices of the edge. Thus upon entering v_0, by enumerating the edges and identifying the marked one, the robot can determine its back-link at v_0. Similar to the case of two undirected V-markers shown in Figure 3.21(a), vertex v_0 in each graph provides explicit back-link information and ambiguous place information – the robot knows it entered one of the two ends of the marked edge but cannot determine which one it is in. Assume that the robot is initially at v_0 and entered via the uniquely marked edge.

Then it can be proved (by induction) that if the robot executes identical motion sequences on the graphs, it would acquire exactly the same perceptions on the different graphs. (The proof is logically similar to the one given for the case of a single undirected marker, and is omitted here for brevity.) The robot thus cannot tell the two different graphs apart.

Given that at an end vertex of a marked edge the robot can determine whether the marker head/tail points to the current vertex, it is straightforward to show that a single directed immovable E-marker provides both explicit place and explicit back-link information at its end vertices (Figure 6.1(b)). Thus, any one of the two end vertices of the marked edge can be used as a directional lighthouse for the algorithm presented in Chapter 3 of the work. Specifically, the single directional immovable V-marker algorithm can then be applied, with the modification that for each hypothesis h', the expected perceptions $\mathcal{P}^E_{h'}$ include the presence or absence of the E-marker at the incident edges of each visited vertex and the direction of the marker if the marker is present. Suppose the end vertex pointed by the marker head is used as the directional lighthouse v_0. Then an example expected perception for hypothesis h' can be expressed as $\mathcal{P}^E_{h'}$=([2,A,A], [2,A,A], [3,A,E-2-H]).

The impoverished algorithm presented in Chapter 5 can also be applied to E-markers. An undirected E-marker, once dropped, provides explicit back-link information at the two end vertices of the edge, and also provides ambiguous place information at the two end vertices – if one end vertex of the marked edge is used as the lighthouse vertex, then only the other end vertex can potentially be confused with the lighthouse. As shown in Figure 6.1(a), this happens only when the two end vertices have the same signatures. Thus we are assured that as long as the two



(a) (b)

Figure 6.2: Graphs with structures can be mapped with an undirected immovable E-marker. (a) An edge with a degree-one end. (b) Any edge whose two ends have different degrees.

ends of the marked edge have different degrees, the place information at the two end vertices are also explicit, and thus any one of them can be used as a directional lighthouse. So, with a single undirected immovable E-marker, the robot can conduct a random walk searching for an edge incident on two vertices with different degrees. Once such an edge is found, the robot can drop the marker in the edge, and designates either one of the end vertices as the directional lighthouse vertex (Figure 6.2), mapping the world deterministically. Note that if the graph is not a *regular* graph, then given sufficient random walk, the robot is able to find a lighthouse vertex, mapping the world deterministically.

6.2 Mapping with multiple immovable markers

Given the high cost of exploring using a single immovable marker, it is interesting to investigate the potential for the increased power of multiple immovable markers in exploration. Assume that all markers are homogeneous. In Chapter 3 it was observed that three undirected immovable V-markers are sufficient to establish direction information and thus solve the SLAM problem deterministically. Three undirected immovable E-markers are also sufficient to solve the problem using a similar strategy: the robot can drop one marker at one of the edges and two markers

at one another edge (assume that v_0 has degree ≥ 2). Then the robot can apply the single immovable directional marker algorithm where the vertex with two marked edges is v_0 (in our model no degenerate edges exist), and the entry edge can be determined based on the relative ordering to one of the marked edges. The final case corresponds to the robot being able to drop markers at both vertices and edges. In this approach two markers are sufficient: one marker is dropped at v_0 and the other marker is dropped along one of its edges, both uniquely marking v_0 and providing explicit back-link information at v_0. Now consider mapping with more markers. Obviously, with more markers the robot can still adopt the hypothesis-based approaches, using two or three markers to establish a directional lighthouse. This approach has $O(m^2n)$ cost bound. But are there efficiencies to be found when more markers are used? This is investigated in this section. This section first considers marker classes consisting of multiple undirected markers, and then looks at the increased power of the marker classes consisting of multiple directional markers. For both undirected and directional markers, first considered is cases in which it is known *a priori* how many markers the robot possesses relative to the size of the world being explored. Then the general case where the relative number of markers is not known was considered. Three different classes of multiple marker algorithms were developed: algorithms that only drop markers at vertices (V-marker algorithms), algorithms that only drop markers on edges (E-marker algorithms) and algorithms that drop markers both at vertices and on edges (V-E-marker algorithms)[15].

[15]Some of the results in this section have been published [64, 65].

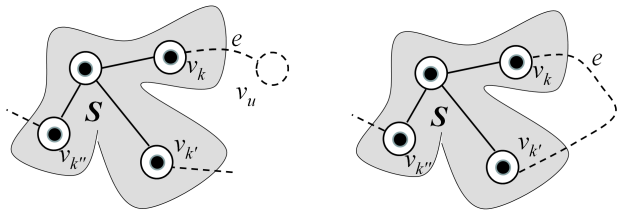


Figure 6.3: Plausible strategy with n markers dropped at each visited vertex. In (b) the robot cannot do further validations. Specifically, it is not able to determine it is arriving at $v_{k'}$ or $v_{k''}$. Both $v_{k'}$ and $v_{k''}$ have the same signature [2,V,A].

6.2.1 Mapping with multiple *undirected* immovable markers

We first consider below several cases in which it is known *a priori* how many markers the robot possesses relative to the size of the world being explored.

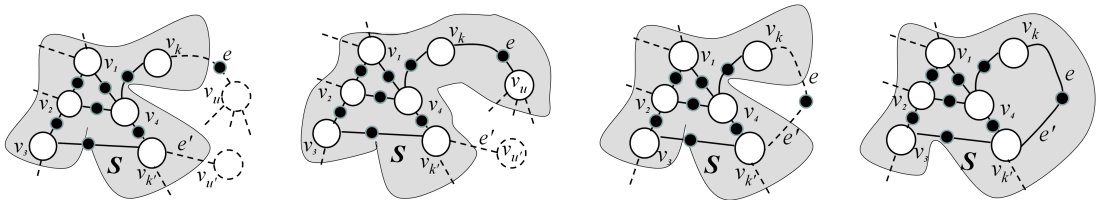
Case 0: Mapping with n markers

First, suppose that the robot is assured that it has at least as many markers as the number of vertices n in the environment. With such an aid, a plausible strategy for the robot is to drop one marker at each vertex as it explores, marking each visited vertex. Since all the visited vertices are marked, the marker sensed at a vertex answers the question 'have I been here before?'. A vertex that contains a marker must be a known (visited) vertex in S, whereas a vertex where no marker is sensed must be a new vertex. When entering a marked vertex, however, the robot may not be able to determine its place and back-link. In the example shown in Figure 6.3(b), the robot cannot determine if it is visiting $v_{k'}$ or $v_{k''}$. Thus this approach does not work in general, and as shown later, other (more complicated) approaches are needed.

Case 1: Mapping with m markers

Suppose that the robot is assured that it has at least as many markers as the number of edges m in the environments. We show two approaches to mapping the world with m markers. We reviewed in Chapter 2 a deterministic algorithm for a robot to map an undirected graph under the *footprints* model [18]. According to the authors, the footprints model can be implemented with $m + n$ homogeneous immovable (undirected) markers, one for each node or edge. The key idea is that when entering a marked place, both place and back-link validations can be conducted simultaneously by searching the known area for the vertex that has a newly marked (foot-printed) edge. Here we show that the same strategy can be adopted using m markers only.

Case 1a: Mapping with m markers, E-marker algorithm We show here that the mapping strategy described in the footprints model can be implemented by dropping markers at each newly visited edge only. The robot drops a marker on newly explored edge e when transiting e. At v_u, the robot senses the marker(s) on the (other) incident edges of v_u. The key observation here is that if v_u has been visited before then it must have at least one other marked edge: if v_u is the initial vertex v_0 then the 'must-be-marked' edge is the incident edge at v_0 along which the robot started the exploration, otherwise the 'must-be-marked' edge is the edge along which the robot first entered the vertex (and hence 'generated' the vertex in S). Also, the robot drops markers only on traversed edges so an unvisited vertex must not have marked edges. Thus if none of the other edges at v_u are marked then v_u must have not been visited before (Figure 6.4(a)) and the algorithm can conduct non-loop augmentation immediately, augmenting S with e (as a marked



(a) v_u has no marked edges

(b) Non-loop augmentation

(c) v_u has marked edges

(d) Loop augmentation

Figure 6.4: Mapping with m undirected E-markers (case 1a). (c) Traverse to a visited place. Sense $c = 2$ marked edge excluding the entry edge. Need to visit $v_{k'}$ and v_1 which have the same degree as v_u and contain c marked edges. (d) $v_{k'}$ is identified, as it has $c + 1 = 3$ marked edges now (e' is the 'unexpectedly marked' edge). S is augmented with $e/e' = (v_k, v_{k'})$ accordingly.

edge) and v_u (Figure 6.4(b)). If v_u has marked edge(s), then v_u must have been visited before (Figure 6.4(c)). In this case v_u and e need to be further validated. As in the footprints model [18], both place and back-link validations are conducted simultaneously by having the robot search the map S looking for the vertex which has one more marked edge than is shown on the map.

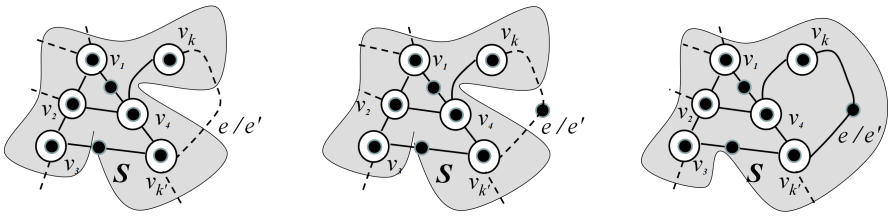
The robot drops one marker on every edge explored, so by the end of exploration a total of m markers are dropped. There are $m - (n - 1)$ iterations of loop augmentation, in which the robot may need to search S. As discussed in [18], in the worst case scenario a search is conducted at each of the $m - (n - 1)$ iterations and each search is bounded by n as the robot may exhaust all the vertices in S when validating a single edge. Thus the exploration cost bound is $O(mn)$.

Case 1b: Mapping with m markers (V-E-marker algorithm) Here we present another way of using m markers to map the world with $O(mn)$ cost. We have discussed in Case 0 that dropping markers on all visited vertices distinguishes visited place and unvisited place and thus allows immediate non-loop augmentation, but does not provide sufficient information for place

and back-link validation. The problem is solved in Case 1a by dropping and searching markers on loop edges. Combining the ideas in Case 0 and Case 1a, this approach drops markers on both vertices and loop edges (hence is a V-E-marker algorithm). Specifically, the markers are dropped at all visited vertices so an unvisited vertex can be determined immediately, and on all the loop edges so the search-based validation can be conducted. Initially the robot drops a marker at its starting vertex v_0. During exploration when the robot enters a vertex v_u via a newly explored edge e, if v_u does not contain a marker then v_u has not been visited before, since all the vertices in S are marked. The robot drops a marker at v_u (but not on the entry edge e), and conducts non-loop augmentation on S with v_u and e (as an unmarked edge). Alternatively if v_u contains a marker then it must have been visited before. Similar to the above m E-marker algorithm, validations are conducted simultaneously by having the robot drop a marker on e and then searching the map S looking for the vertex that has one more marked edge than what is shown in S. The algorithm is shown in Figure 6.5. The robot drops one marker at each vertex it visits, so at the end of the algorithm the robot has dropped n markers at the vertices of the graph. The robot also drops a marker on each of the $m-(n-1)$ loop edges. So in total $m+1$ markers are dropped[16]. Same as the previous algorithm, this algorithm has $O(mn)$ cost bound.

This $m+1$ marker algorithm can be extended slightly so that it works under the original assumption that the robot is assured that it has at least m markers. This algorithm uses one marker in each iteration, so if the robot runs out of markers during exploration, then it must be

[16]Another way of showing that $m+1$ are used is to observe that initially one marker is dropped at the starting vertex v_0, and then in each of the m iterations exactly 1 marker is dropped (either on the vertex or on the edge, depending on whether v_u is a marked place or not).



(a) v_u has marked edges (b) Drop a marker at e and search S (c) Loop augmentation

Figure 6.5: Mapping with $m+1$ undirected V-E-markers (case 1b). (a) Traverse to a marked place. Sense $c = 1$ marked edge. (b) Return to v_k, dropping a marker on e. Need to visit $v_{k'}$ and v_1, which have the same degree as v_u and contain c marked edges. (c) $v_{k'}$ is identified, as it has $c + 1 = 2$ marked edges now (e' is the 'unexpectedly marked' edge).

true that only one iteration of the algorithm execution remains. Now either U has one unexplored non-loop edge in it, which would lead to an unmarked vertex that has no other incident edges, or, U has two unexplored loop edges in it and they correspond to the same edge (a loop is formed). So the modified algorithm just executes the $m+1$ marker algorithm, but if the robot runs out of markers and U is not empty yet (one or two edges in U), it just processes the edge(s) in U accordingly without further motion.

Case 2: Mapping with $m+n$ markers (E-marker algorithm)

Assume that the robot is assured that it has at least $m+n$ undirected markers. We can use the markers to implement the footprints model algorithms but we have shown in Case 1 that m markers are sufficient. Are there efficiencies to be found with $m+n$ markers?

In the two algorithms in Case 1, when exploring to a marked place v_u, the number of marked edges c at v_u is used to select potential candidate vertices for v_u that need to be searched. In the E-marker algorithm where each visited edge is marked, c represents the current number of

explored edges at v_u. In the V-E marker algorithm where only loop edges are marked, c represents the current number of explored loop edges at v_u (but not the total number of explored edges). The idea here is to combine the information used in these algorithms, i.e., exploit both the total number of explored edges and the number of explored loop edges. This enriched information, which distinguishes the two types of explored edges (loop and non-loop) at v_u, may lead to a reduced number of potential candidates that need to be visited and thus potentially reduces the search cost. With $m + n$ markers, we extend the m E-marker algorithm by dropping one marker on each loop edge as before (call them single-marked edges), but two markers on each non-loop edge (call them double-marked edges). Thus at each marked place v_u, the robot can, in addition to the total number of marked edge, sense both the number of (single-marked) loop edges c_1 and (double-marked) non-loop edges c_2 ($c_1 + c_2$ is the totally number of explored edges at v_u). Then the robot uses this information to select the potential candidate for v_u. A known vertex in S is a potential candidate if it has c_1 single-marked edges and c_2 double-marked edges. In this algorithm $m + n$ markers is sufficient to maintain the enriched information: the robot drops $m - n + 1$ markers on loop edges and drops $2(n - 1)$ markers on the $n - 1$ non-loop edges, so totally $m + n - 1$ markers are required in the algorithm. Note that this algorithm has the same $O(mn)$ cost bound as the previous two m marker algorithms. The real cost is expected to be lower than the previous two algorithms due to the potentially reduced search efforts. The algorithm is shown in Algorithm 6.1.

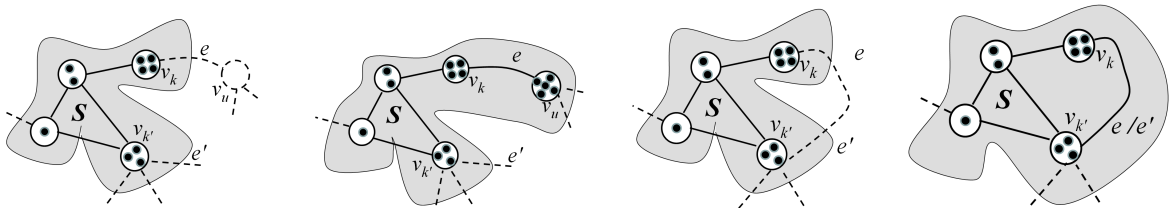
Algorithm 6.1: Mapping with $m+n$ undirected immovable markers (Case 2, E-marker)

```
1  S ← v_0; U ← incident edges in v_0;
2  while U is not empty do
3      remove an unexplored edge e = (v_k, v_u) from U;
4      the robot traverses S to v_k and then follows e to v_u;
5      if v_u has no marked edge then
           // v_u has not been visited.  do non-loop augmentation;
6          the robot drops two markers on e;
7          add e to S as a double-marked edge, add v_u to S, add other edges in v_u to U;
8      else // v_u contains, say, c_1 single-marked edge, c_2 double-marked edge;
9          the robot goes back to v_k via e, drops a marker during transit on e;
10         for each other vertex in S that has: 1) unexplored edge(s) 2) same degree as v_u 3)
           c_1 single-marked edges 4) c_2 double-marked edges do
11             robot goes there, sensing the marked edges there;
12             if a vertex called v_k' has c_1 + 1 single-marked edges then
                   // do loop augmentation;
13                 identify the unexpectedly single-marked edge e' of v_k', based on the relative
                   ordering to the known entry edge;
14                 add e/e' = (v_k, v_k') to S as a single-marked edge; remove e' from U;
15                 exit for loop // stop searching;
16 return S;
```

Case 3: Mapping with $\sum_{k=1}^{n-1} k$ markers

Assume that the robot is assured that it has at least $\sum_{k=1}^{n-1} k = n(n-1)/2$ undirected markers. We present two approaches to using the markers in exploration.

Case 3a: Mapping with $\sum_{k=1}^{n-1} k$ markers (V-marker algorithm) We present here a simple V-marker algorithm that uses $\sum_{k=1}^{n-1} k$ markers that are dropped on vertices and has a $O(md_{max})$ cost bound, where d_{max} is the maximum vertex degree in the graph. In this approach, the robot drops a different number of markers at each vertex as it explores, marking each visited vertex uniquely. Every visited vertex has a unique signature [degree, V#, A] in which '#' is a



(a) v_u is not marked

(b) Non-loop augmentation by e and v_u

(c) v_u is marked

(d) Loop augmentation of S by e/e'

Figure 6.6: Mapping with $\sum_{k=1}^{n} k$ undirected V-markers. (c) identify the explored edges at $v_{k'}$, generating a set $s = \{1, 2\}$ representing other explored edges. Traverse each (other) edges at $v_{k'}$. (d) stop when sensing either one or two markers at the other end vertex.

unique number. The unique signature at each visited vertex not only answers 'have this vertex been visited before?' but also answers 'exactly which vertex the currently visited vertex is?'. Thus explicit place information is provided in each marked vertex. When exploring to an unmarked vertex v_u via e, both v_u and e are added to S (non-loop augmentation) immediately. The unique number of markers contained in v_u is also recorded. When exploring to a marked vertex v_u, based on the unique signature of the vertex, the robot knows immediately which vertex in S it is visiting (call it $v_{k'}$). However, explicit back-link information may not exist and thus the robot may need to perform 'back-link validation' (see Figure 6.6(c) for an example). The key observation here is that if $v_{k'}$ has been visited before then it must have at least one explored edge in S, whose other end is also known. Thus the back-link validation problem can be solved by determining the relative edge ordering between the entry edge and one of the explored edges at $v_{k'}$. Given that no degenerate edges exist in the world, the problem can be reduced to looking for a known neighbor of $v_{k'}$. The algorithm retrieves from S all the explored edges at $v_{k'}$, and generates a set s of other end vertices of the explored edges, represented by the unique number

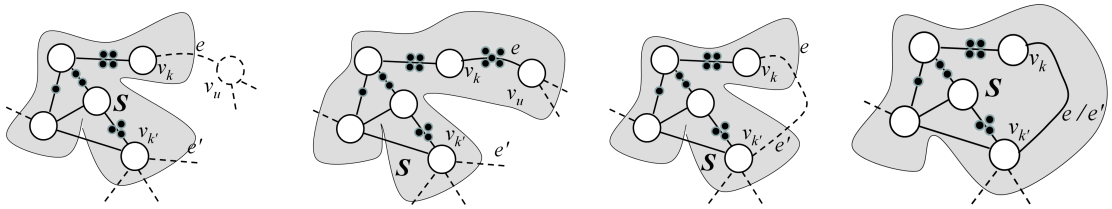
of markers they contain. Then the robot traverses each (other) incident edge at $v_{k'}$ to the other end vertex, until it has sensed c markers where $c \in s$.

There are $m - (n-1)$ iterations during which the robot explores to a marked vertex and needs to conduct back-link validation. In the worst case scenario extra traversals are required in each of the iterations, and all other incident edges at the vertex need to be traversed. Thus the cost of the algorithm is bounded by $O(md_{max})$ where d_{max} is the maximum vertex degree in the graph. The algorithm marks each of the n vertices uniquely, so a total of $\sum_{k=1}^{n} k$ V-markers are used. This algorithm can be modified so that it works under the original assumption that the robot is assured $\sum_{k=1}^{n-1} k$ markers. With $\sum_{k=1}^{n-1} k$ markers the robot can mark $n-1$ vertices. So if the robot enters an unmarked place but for the *first* time it does not have sufficient markers to mark the place uniquely then this vertex must be the last vertex that has not been visited and the robot can leave the vertex blank. From then on whenever the robot enters an unmarked vertex, the vertex is identified as the spacial known vertex based on its unique blank label. This version of the algorithm is sketched in Algorithm 6.2.

Case 3b: Mapping with $\sum_{k=1}^{n-1} k$ markers (E-marker algorithm) With $\sum_{k=1}^{n-1} k$ markers that are dropped at vertices, the robot can map the world with $O(md_{max})$ cost bound. Here we present another algorithm, which uses the same number of markers but achieves $O(m)$ cost bound. This is an E-marker algorithm, in which the robot drops different number of markers at edges that lead it to unvisited places, thus labelling each *non-loop* edge uniquely. Loop edges and vertices are not marked. Given that no degenerate edges exist, the key observation is that by marking each non-loop edge uniquely as the robot explores, the robot uniquely labels each visited

Algorithm 6.2: Mapping with $\sum_{k=1}^{n-1} k$ undirected markers (Case 3a, V-marker algorithm)

```
1  i ← 1; // number of markers to drop next;
2  the robot drops i marker at v_0;
3  S ← v_0; U ← edges in v_0;
4  visitedAllVertices = False; // whether all the vertices have been explored;
5  while U is not empty do
6      remove an unexplored edge e = (v_k, v_u) from U;
7      the robot traverses S to v_k and follows e to v_u;
8      if v_u does not contain marker(s) AND not visitedAllVertices then
           // v_u is a new place;
9          if the robot has enough markers to mark v_u then
10             i ← i + 1 ;
11             the robot drops i markers at v_u;
12             do 'non-loop augmentation' on S by e and v_u (as a i-marked vertex);
13         else // does not have enough markers.  Leave it blank.
14             do 'non-loop augmentation' on S by e and v_u (as an unmarked vertex);
15             visitedAllVertices = True;
16     else // v_u contains markers OR (no marker but) vistedAllVertices==true
17         identify v_u from S based on the unique number of markers at v_u, say, v_k';
18         identify from S all explored edges at v_k', generating a set s of other end vertices,
           represented by the unique number of markers at each end vertex;
19         for each other edge incident in v_k' do
20             the robot traverses to the other end, sensing the markers there;
21             if c ≥ 1 markers are sensed OR (no marker is sensed AND
               vistedAllVertices==true) then
22                 if c ∈ s then
                       // traversed an explored edge;
23                     identify the edge from S based on c, say, e'';
24                     based on the relative ordering to e'', identify entry edge e';
25                     do loop augmentation on S with edge e/e' = (v_k, v_k');
26                     remove e' from U;
27                     exit for loop;
28                 else // c ∉ s.  not an explored edge.
29                     return to v_k';
30             else // no marker is sensed AND visitedAllvertices==false.
31                 return to v_k'; // not an explored edge.
32 return S;
```



(a) v_u has no marked edges

(b) Non-loop augmentation

(c) v_u has marked edges

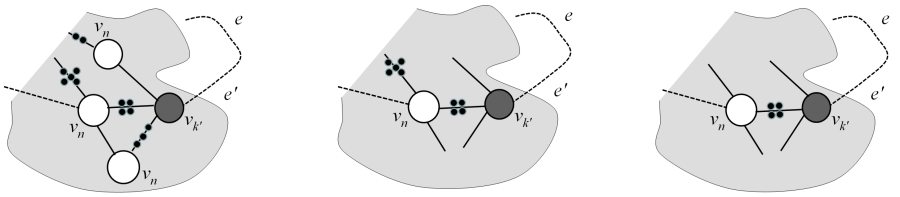
(d) Loop augmentation of S by e/e'

Figure 6.7: Mapping with $\sum_{k=1}^{n-1} k$ undirected E-markers (case 3b). In (c) $v_{k'}$ is uniquely identified.

place and edges. Specifically, it is maintained true that (1) a vertex has marked edge(s) if and only if it has been explored before. (2) if v_u has marked edge(s), then both v_u and its incident edges can be uniquely identified. That is, both explicit place and back-link information exist in v_u. We first present an algorithm based on this observation, and then justify the observation.

Upon entering an unknown vertex v_u, the robot senses the marker(s) on the incident edges of v_u. If none of the edges at v_u are marked then according to (1), the algorithm conducts non-loop augmentation on S with e and v_u immediately. Edge e is a non-loop edge. In order to maintain that each non-loop edge is marked, the robot goes back to (non-loop) edge e, dropping a unique number of markers on e. If, alternatively, one or more edges of v_u are marked, then according to (2) we can then infer the entry edge e' by enumerating the edges and identifying the marked ones. Physical motion for both place and back-link validations is avoided. Thus the approach behaves similar to search algorithms such as Depth-first search (DFS), which has $O(m)$ cost. The algorithm is illustrated in Figure 6.7. The robot drops markers on non-loop edges, but not on loop edges. There are $n-1$ iterations of non-loop exploration. The total number of markers required thus is $\sum_{k=1}^{n-1} k$.

Here we justify the key observation based on which the algorithm is developed. In the algorithm the robot drops markers on each non-loop edge. This creates a marked spanning tree on S. So if v_u has been visited before then it must contain at least one marked edge. Moreover, the robot only drops markers on traversed edges, so an unvisited vertex must not have marked edges. Thus if v_u has marked edge(s) then it must have been visited before, otherwise it must be a new vertex. This justifies observation (1). Supposing v_u contains marked edge(s), we further justify observation (2) that its E-markers uniquely identify the vertex as well as its incident edges. Suppose v_u corresponds to a known vertex $v_{k'}$. We show that there exist no vertices in S that are potentially confusing to $v_{k'}$. Given that each of the vertices in S has at least one uniquely marked edges, it is trivially true that vertices in S that are not known neighbors of $v_{k'}$ are not potentially confusing with $v_{k'}$, as each of such vertices has its unique edge(s) that is distinct from the marked edge(s) of $v_{k'}$. A known neighbor of $v_{k'}$ connects with $v_{k'}$ either via an unmarked edge or a marked edge. A neighbor that connects $v_{k'}$ via an unmarked edge is also trivially distinguishable from $v_{k'}$ as it must have at least one marked edge that is distinct from the marked edges of $v_{k'}$. Now consider a neighbor vertex v_n that connects $v_{k'}$ by a marked edge. The key observation here is that now either v_n or $v_{k'}$ or both must have other uniquely marked edge(s), so that they are distinguishable to each other, as shown in Figure 6.8(a) and (b). The situation that both $v_{k'}$ and v_n have only one marked edge, which is the edge connecting them, as shown in Figure 6.8(c), can *not* happen by construction. The markers on the (non-loop) edge connecting $v_{k'}$ and its neighbor v_n indicate that the two vertices have a 'parent-child' relation in the marked spanning tree. Consider two possibilities. If the parent node is not the root node



(a) $v_{k'}$ has 2 marked edges.

(b) $v_{k'}$ has 1 marked edge, neighbor has 2.

(c) $v_{k'}$ and neighbor has 1 marked edge.

Figure 6.8: Mapping with $\sum_{k=1}^{n-1} k$ undirected E-markers. Explore to a visited place $v_{k'}$. $v_{k'}$ can be uniquely identified in (a) and (b). In (a) $v_{k'}$ has unique signature [4,A,E3-1,E4-2]. In (b) $v_{k'}$ has unique signature [4,A,E4-2]. In (c) entering $v_{k'}$ and its neighbor v_n generate the same signature [4,A,E4-2], but this situation can *not* happen in the algorithm.

of the spanning tree, then clearly the parent node must have a non-loop edge connecting to its parent node. Alternatively if the parent node is the root node, then the fact that a loop is formed at $v_{k'}$ requires that either the parent (root) node has at least one more child node, or the child node has at least one child node of its own. In both cases, at least one of v_u or v_n should have at least one more marked edge. This justifies observation (2), which implies that v_u can be uniquely identified by its marked edge(s). Once v_u is identified, the entry edge e' can be identified based on the relative ordering between e' and one of the marked edges, whose index is known in S.

Case 4: Mapping with $m - \lfloor \frac{n}{2} \rfloor$ and m markers – extensions to the two $O(mn)$ E-marker algorithms

We presented above three search-based algorithms which are based on the footprints model algorithm given in [18] and map the world with $O(mn)$ cost bound. Two of them use m markers (Case 1a which is an E-marker algorithm and Case 1b which is a V-E marker algorithm). The other algorithm (Case 2) uses $m + n$ markers, which is an E-marker algorithm. Here we present

extensions to the two E-marker algorithms. The extensions have the same $O(mn)$ cost bound but reduce the number of markers used from m to no more than $m - \lfloor \frac{n}{2} \rfloor$ markers, and from $m + n$ to no more than m markers.

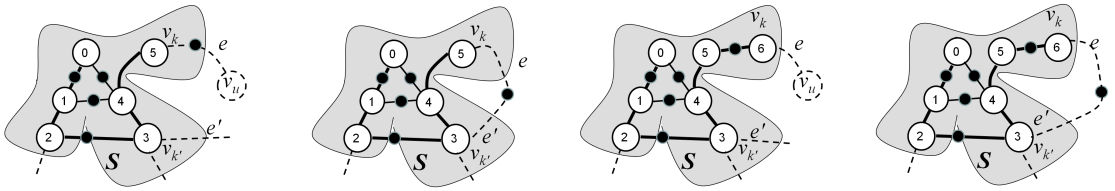
The extensions strive to save some markers used in the previous algorithms, while maintaining true the property that a vertex has marked edge(s) if and only if it has been visited. The idea is to drop markers on all loop edges (as in the algorithms) but drop markers only on *some* of the non-loop edges. The fact that not all the non-loop edges are marked reduces the number of marker required, but also necessitates extra efforts in order to maintain that each visited vertex has marked edge(s). In particular, the robot needs to explore in some pattern and here we show one of the possible patterns. We enforce that the robot selects from U the edge e whose known end v_k is the *closest* vertex to the current vertex – rather than an arbitrary edge in U. First consider extending the m marker algorithm described in Case 1a where one marker is dropped at each explored edges. In this extension, the robot follows the following rules during exploration:

(1) chooses edge $e = (v_k, v_u)$ where v_k is the closest known place to the current place of robot (measured in terms of the number of edge traversals required).

(2) if e is a loop edge (i.e., v_u has been visited before), then a marker is (always) dropped on e – this marker is needed to do place and back-link validations as in the original algorithm.

(3) if e is a non-loop edge (i.e., v_u is a new vertex), then a marker is dropped on e *only* if v_k has no marked edge yet – so v_k has a marked edge now.

Following these rules ensures that when the robot enters a vertex v_u that has been visited before,



(a) v_k has no marked edges. e is non-loop edge

(b) v_k has no marked edges. e is loop edge

(c) v_k has marked edges. e is non-loop edge

(d) v_k has marked edges. e is loop edge

Figure 6.9: Extension to m E-marker algorithm shown in Case 1a. Numbers in vertices show the order the vertices are explored. If v_k has no marked edges, e is marked, as in (a)(b). If v_k has marked edges, e is marked only if it is a loop edge, as in (d). e is not marked in (c).

the vertex must have at least one marked edge. Consider the known end v_k of a newly explored edge $e = (v_k, v_u)$. Case 1: v_k has no marked edge yet at the begining of the current exploration iteration (e.g., the very first iteration in which v_0 is v_k). After the current exploration, $e = (v_k, v_u)$ is either a loop edge or a non-loop edge but e is marked in either case (according to rules (2) and (3)), as shown in Figure 6.9(a)-(b). Now v_k has a marked edge now. The other end v_u, which is a visited vertex now, also has at least one marked edge. Case 2: v_k has marked edge(s) at the beginning of the current iteration (e.g., the iteration that follows case 1, with v_u in case 1 being v_k now). If e turns out to be a loop edge, then e is marked according to rule (2), resulting in v_k and v_u having one (more) marked edge. If, alternatively, e is an non-loop edge, then according to rule (3) no marker is dropped at e. This results in v_u, which is a visited vertex now, still having no marked edges. Now if the robot continues the next iteration of exploration from an arbitrary unexpored edge in U then the robot might enter v_u again (in the next or later iterations), where it might not see any markers, although v_u is an explored vertex. In order to avoid this problem, rule (1) enforces that the robot selects the *closest* edge to explore in the next iteration. Since the

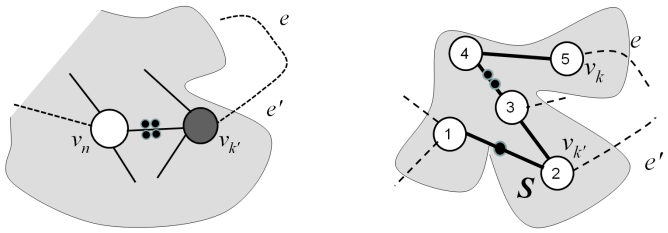
robot is at v_u at the end of current iteration, this ensures that the robot selects one unexplored edge at v_u to explore next (i.e., the robot explores 'depth-first'). This edge will be marked in the next iteration – regardless of whether the edge turns out to be a loop edge or an non-loop edge (according to rule (2) and (3)). Note that other exploration patterns that avoid this problem can also be applied. The robot only drops markers on traversed edges thus an unvisited vertex has no marked edge. Thus it is maintained true that a vertex contains marked edges only if it has been visited before. The algorithm is sketched in Algorithm 6.3.

The algorithm drops a marker at each of the $m-(n-1)$ loop edges, and drops a marker at a non-loop edge $e=(v_k,v_u)$ only if v_k has no marked edge. Maximumly, half ($\lceil \frac{n-1}{2} \rceil$) of the non-loop edges are marked (consider a world of a chain or a cycle, where every other non-loop edge is marked). Totally $m-\lfloor \frac{n}{2} \rfloor$ markers are sufficient. So by enforcing an exploration pattern, this extension reduces the number of markers used in the original algorithm from m to $m-\lfloor \frac{n}{2} \rfloor$.

We can apply the same technique to extend the $m+n$ E-marker algorithm, which drops two markers at each non-loop edge and one marker at each loop edge. With the same technique, the robot explores in a closest-first manner, and drops (two) markers on non-loop edges $e=(v_k,v_u)$ only if the known end v_k contains no marked edges yet. In this extension, the robot drops $m-n+1$ markers on loop edges as before, and drops a maximum of $n-1$ markers on half of the non-loop edges. So m markers are sufficient. So this extension reduces the number of markers used in the original algorithm from $m+n$ to m. This extension is another way of using m markers, and is expected to provide reduced cost than the other two m marker algorithms, since this extension uses both c_1 and c_2 in selecting potentially confusing vertices to visit.

Algorithm 6.3: Mapping with $m - \lfloor \frac{n}{2} \rfloor$ undirected E-markers (Case 4 an extension)

```
1  S ← v_0; U ← incident edges in v_0;
2  while U is not empty do
3      remove an unexplored edge e = (v_k, v_u) from U where v_k is the closest vertex;
4      the robot traverses S to v_k and then follows e to v_u;
5      if v_k has no marked edge(s) then
6          the robot drops a marker at e during transit on e;
           // if v_k has marked edges, leave e blank;
7      the robot senses marked edges at v_u (excluding entry edge e);
8      if v_u has no other marked edge then
           // v_u has not been visited.  do non-loop augmentation;
9          if a marker was just dropped on e then
10             add e to S as a marked edge;
11         else
12             add e to S as an unmarked edge;
13         add v_u to S; add other edges in v_u to U;
14     else // v_u contains, say, c marked edge(s) excluding entry edge;
15         the robot returns to v_k via e;
16         if no marker has been dropped at e then
17             the robot drops a marker at e during transit on e; // loop-edge marked;
18         for each vertex in S that has: 1) unexplored edge(s) 2) same degree as v_u 3) c
           marked edges do
19             robot goes there, sensing the marked edges there;
20             if a vertex called v_k' has c + 1 marked edges then
21                 identify the unexpectedly marked edge e' of v_k', based on relative ordering
                   to the known entry edge;
22                 add e/e' = (v_k, v_k') to S as a marked edge; // loop-augmentation;
23                 remove e' from U;
24                 exit for loop;
25 return S;
```



(a) $v_{k'}$ and neighbor share one marked edge.

(b) $v_{k'}$ and neighbor can be confused.

Figure 6.10: Trying to reduce markers in $\sum_{k=1}^{n-1} k$ E-markers algorithm. In (b) entering $v_{k'}$ (node 2) and its neighbor (node 1) might generate the same signature [3,A,E1-1].

Can we apply the same idea to the $\sum_{k=1}^{n-1} k$ algorithms presented earlier? Consider the $\sum_{k=1}^{n-1} k$ E-marker algorithm described in Case 3b. This algorithm drops markers on all non-loop edges (only). These uniquely marked non-loop edges at a visited place identify the place and the incident edges so $O(m)$ cost is maintained. Can we save markers by dropping markers on a non-loop edge $e = (v_k, v_u)$ only if v_k has no marked edge yet? Unfortunately, these marked edges may not always identify a visited vertex and its incident edges. That is, the situation shown in Figure 6.8(c) and repeated here in Figure 6.10(a), which is impossible in the original algorithm, can now happen. In this case, $v_{k'}$ and neighbor v_n may not always be distinguishable. A scenario is shown in Figure 6.10(b). We revisit this topic when we investigate multiple *directional* markers.

Case 5: Mapping with arbitrary $k > 3$ undirected markers

Above we considered several cases where the robot has prior knowledge about the relative number of markers it possesses. Now consider the more general (and more challenging) problem where the robot does not know *a priori* the relative number of markers. The robot either does not know

the graph size, or knows the graph size but does not have enough markers to apply the above approaches. As discussed earlier, probably the simplest approach is to establish a directional lighthouse using two or three markers, which are dropped at the initial location and one of its edges or neighbors, and then run the single directional marker algorithm with $O(m^2n)$ cost bound. But when we have more than three markers, can we do the exploration more efficiently?

Possible approaches must consider both the case where the robot has sufficient markers and also the case where the robot runs out of markers during exploration. The robot must be able to conduct validations in both cases. So the possible approaches proceed in two phases. In phase-I the robot uses markers to do validations, and in phase-II the robot conducts exploration using the markers placed in phase-I. We have developed above several algorithms in which the robot uses markers to do validations. These algorithms can potentially be used in phase-I. When the robot runs out of its markers, we apply the hypothesis-based traversal strategy developed for the single immovable marker algorithm. In order to do this, in phase-II the robot must be able to select from S a directional lighthouse vertex. So the search-based algorithms described in Case 1 and Case 2, which may maintain a 'homogeneously marked' S, can not be used directly in phase-I of exploration. We can either slightly modify these algorithms so that at least one known vertex or edge in S is marked uniquely and also provides back-link information, which can be used as the directional lighthouse in phase-II, or apply one of the two $\sum_{k=1}^{n-1} k$ algorithms (Case 3) where vertices or edges in S are marked uniquely. We present here both the approaches.

Version-1 The first approach is a V-marker algorithm that combines the techniques used in the $\sum_{k=1}^{n-1} k$ V-marker algorithm (Case 3a, $O(md_{max})$ cost bound) and the single directional

V-marker algorithm (Algorithm 3.1). A similar E-marker algorithm can be developed using the technique in the $\sum_{k=1}^{n-1} k$ E-marker algorithm (Case 3b) and the single directional E-marker algorithm. In phase-I, the robot marks each newly visited vertex v_u by dropping a different number of V-markers. As in the $\sum_{k=1}^{n-1} k$ V-marker algorithm, an unmarked place is a new place. When entering a marked place v_u, the unique marker count identifies v_u but the robot may need to conduct back-link validation by traversing each other incident edges until a known neighbor is visited. When the robot comes to an unmarked vertex but does not have sufficient markers to uniquely mark the vertex, phase-I ends and the algorithm enters phase-II.

A different validation strategy is used in pahse-II. S may contain both marked and unmarked vertices. Upon entering a marked vertex v_u, the robot conducts back-link validation as in phase-I. If v_u is unmarked, then it could be a new place but could also be a known place explored in phase-II. The robot conducts a hypothesis-based traversal similar to that in the single directional V-marker algorithm (Algorithm 3.1). Specifically, each unexplored edge $e' = (v_{k'}, v_{u'})$ and its known end $v_{k'}$ is a possible loop closing hypothesis h' if its known end $v_{k'}$ has the same signature as v_u, i.e., $v_{k'}$ is *unmarked* and has the same degree as v_u. To validate the hypothesis, a *marked* (known) place v_l is chosen as the lighthouse vertex for validating the hypothesis. Unlike in the single directional marker algorithm where only v_0 can be used a lighthouse, here any of the marked places can be used as a lighthouse. To reduce exploration cost, we choose the marked vertex that is *closest* to $v_{k'}$ (in terms of edge traversals) as the lighthouse. The back-link at v_l is determined based on the edge connecting to one of the marked neighbors (there must be at least one such neighbor). Spacifically, a simple motion sequence $\mathcal{M}_{h'}$, which drives the robot from $v_{k'}$

to v_l and then to a known neighbor v_n without repeated vertices, is computed. The expected perception $\mathcal{P}^E_{h'}$ is also computed, which includes (at the end) the signatures of v_l and v_n. $\mathcal{M}_{h'}$ along with $\mathcal{P}^E_{h'}$ defines an embedded path $v_l, ..., v_{k'}, v_{u'}$ on S, which uniquely identifies h'.

The algorithm is outlined in Algorithm 6.4. The algorithm has the same $O(m^2n)$ cost bound, while in practice it is expected to produce reduced cost over the single undirected marker algorithm due to the reduced need to conduct hypothesis validation (only in phase-II), reduced number of potential hypothesis (only edges incident on *unmarked* known end), and the reduced number of edge traversals in path executions (lighthouse is the *closest* vertex). Note that when the robot has sufficient markers to mark each visited vertex, the algorithm behaves as the $\sum_{k=1}^{n-1} k$ V-marker algorithm which has $O(md_{max})$ cost bound.

Version-2 This alternative approach applies a modified search-based algorithms in phase-I. In the following we demonstrate applying the approach in the m V-E-marker algorithm in phase-I. A similar approach is possible by applying the m E-marker algorithm or the $m + n$ E-marker algorithm and the extensions of them. The m V-E-marker algorithm cannot be applied in phase-I directly as it maintains homogeneously marked vertices in S but in phase-II the robot needs at least one unique place as the directional lighthouse place. In order to create a directional lighthouse for exploration in phase-II, we modify the m V-E-marker algorithm as follows. Initially the robot drops two markers at its initially location v_0, and then drops two markers at one of the incident edges of v_0. Then the subsequent steps are very similar to the original m V-E-marker algorithm. Specifically, during exploration the robot drops a marker at each visited place if it is unmarked. Thus each known vertex in S contains either two or one markers. When the robot

Algorithm 6.4: Mapping with $k > 3$ undirected markers. Version-1 (V-marker algorithm)

```
1  the robot drops a marker at v_0;
2  S ← v_0; U ← edges in v_0;
3  i ← 1;
4  while U is not empty do
5      remove an unexplored edge e = (v_k, v_u) from U;
6      the robot traverses S to v_k and follows e to v_u;
7      if v_u contains marker(s) then
8          the robot does 'back-link validation' by traversing other incident edges at v_u until
           an explored edge has been traversed – see Algorithm 6.2;
9          do 'loop augmentation' on S; remove entry edge e' from U;
10     else // v_u is unmarked
11         if the robot has at least i markers then
12             the robot drops i markers at v_u;
13             do 'non-loop augmentation' on S with e and v_u (as an i-marked vertex);
14             i ← i + 1;
15         else // not sufficient markers to mark v_u, phase II
16             H ← set of edges in U and known ends whose known ends have same signature
               as v_u, i.e., the known ends are unmarked and have the same degree as v_u;
17             while H is not empty do
18                 h' = (e', v_k') ← a hypothesis removed from H;
19                 choose as lighthouse vertex a marked vertex v_l that is closest to v_k';
20                 choose a marked neighbor of v_l, call it v_n;
21                 compute a simple motion sequence M_h' which drives the robot from v_k' to
                   v_l then to v_n, and the expected perception P^E_h';
22                 the robot attempts to execute M_h';
23                 based on the perception info P_h' obtained in executing M_h' do
24                     case P_h' and P^E_h' do not match throughout
25                         reject the hypothesis, back to v_u
26                     case P_h' and P^E_h' match throughout
27                         confirm hypothesis, exit inner while loop;
28             if a hypothesis e' = (v_k', v_u') is confirmed then
29                 do 'loop augmentation' on S with e/e' = (v_k, v_k');
30                 remove e' from U;
31             else // all hypotheses are rejected
32                 do 'non-loop augmentation' on S with e and v_u (as an unmarked vertex);
33 return S;
```

comes to marked vertex v_u, if v_u contains one marker, then as in the original algorithm the robot drops a marker on e and then searches S for the vertex which now has an 'unexpectedly marked edge'. In phase-II, each visited vertex in S contains either two, one or zero markers. If v_u contains no marker, then it could be a new vertex or could be a known vertex explored in phase-II. If v_u contains one marker, then it is a known vertex explored in phase-I. In both cases we use the hypothesis-based approach to do place and back-link validation. Except when exploring to v_0 where two markers are sensed, at each unknown place v_u (which contains zero or one marker), the robot uses v_0 as the directional lighthouse vertex to do hypothesis-based validation, where the entry edge can be examined by the robot traversing edges in v_0 looking for the edge with two markers. One version of the approach is shown in Algorithm 6.5. Note that when the robot has sufficient markers to mark each vertex and loop edge, this algorithm behaves like the m V-E-marker algorithm which has $O(mn)$ cost bound.

Empirical evaluations of exploring with multiple undirected markers

The performance of mapping with various classes of multiple undirected immovable markers discussed above are evaluated empirically. We first examine mapping with different known classes of multiple immovable markers. These classes include the m E-marker algorithm, m-V-E marker algorithm and $m + n$ E-marker algorithm, which both have $O(mn)$ cost bound. These also include the $\sum_{k=1}^{n-1} k$ V-marker algorithm which has $O(md_{max})$ cost bound and the $\sum_{k=1}^{n-1} k$ E-marker algorithm which has optimal $O(m)$ cost bound. We also examine the two extensions to the two $O(mn)$ E-marker algorithms. Based on the nature of the algorithms and their cost

Algorithm 6.5: Mapping with k undirected markers. Version-2 (V-E-markers algorithm)

```
1  the robot drops two markers at v_0, and two markers at one of the incident edges at v_0;
2  S ← v_0; U ← edges in v_0;
3  while U is not empty do
4      remove e = (v_k, v_u) from U; the robot traverses S to v_k and follows e to v_u;
5      if v_u contains two markers then
6          validate back-link by identifying the edge containing two markers;
7          do 'loop augmentation' on S;
8      else if v_u contains one marker then
9          if the robot has one or more markers then
10             does 'back-link validation' by sensing number of marked edges, say c, and back
               to v_k via e, dropping a marker at e, and then searching S for the vertex that
               has c + 1 marked edges. See algorithms in Case 1;
11             do 'loop augmentation' on S; remove entry edge e' from U;
12         else // run out of markers.  Phase-II
13             H ← set of edges in U and their known ends where each known end contains
               one marker and has the same degree as v_u;
14             while H is not empty do
15                 h' = (e', v_k') ← a hypothesis removed from H;
16                 the robot attempts to execute a computed simple motion sequence M_h';
17                 based on the perceptions P_h' obtained in executing M_h' do
18                     case P_h' and P^E_h' do not match throughout
19                         reject the hypothesis, continue;
20                     case P_h' and P^E_h' do not match throughout
21                         confirm hypothesis, exit for loop;
22             if a hypothesis e' is confirmed then
23                 do 'loop augmentation' on S; remove e' from U;
24             else // no hypothesis exists, or all hypotheses are rejected
25                 do 'non-loop augmentation' on S and U;
26     else
           // v_u contains no marker;
27         if the robot has one or more markers then
28             the robot drops a marker at v_u;
29             do 'non-loop augmentation' on S with v_u as a marked vertex;
30         else // run out of markers.  Phase-II
31             H ← set of edges in U and known ends where each known end contains no
               marker and has the same degree as v_u;
32             conduct hypothesis-based validation as above (step 14 – 26)
33 return S;
```

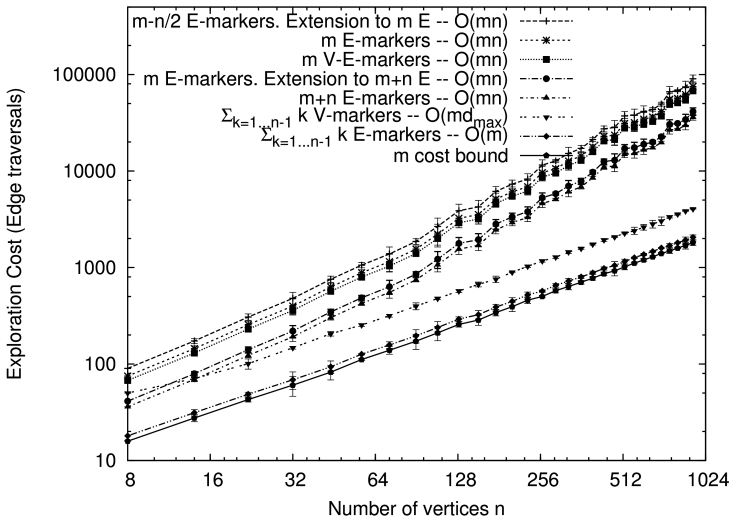
bounds, exploring with m E-markers and m V-markers are expected to have similar costs, and exploring with $m + n$ E-markers, which have the same $O(mn)$ cost bound, is expected to have lower cost. Exploring with $\sum_{k=1}^{n-1} k$ E-markers which has $O(m)$ cost bound, should have the lowest cost. It is interesting to examine the relative performance of the $\sum_{k=1}^{n-1} k$ V-marker algorithm which has $O(md_{max})$ cost bound. It is also interesting to examine the two extension algorithms, both their exploration costs and their marker consumptions.

The different algorithms are tested on a similar set of graphs as used before – both lattice-like graphs and densely connected graphs with randomly deleted edges. Results are shown in Figure 6.11. For comparison purposes, the theoretical lower cost m is also plotted. Among the several algorithms that have $O(mn)$ cost bounds, as expected, the m E-marker algorithm and the m V-E-marker algorithm have slightly different but very similar costs. The $m+n$ E-marker algorithm, in which extra markers are used to distinguish loop and non-loop edges, provides lower cost. The $\sum_{k=1}^{n-1} k$ E-marker algorithm, which has $O(m)$ cost bound, provides the lowest cost, which is very close to the optimal cost m. The $\sum_{k=1}^{n-1} k$ V-marker algorithm which has $O(md_{max})$ cost bound has different relative performance in the two types of environments. In lattice graphs where d_{max} is a small number (≤ 4), the cost of the algorithm lies between the costs of $O(mn)$ algorithms and the $O(m)$ algorithm. In densely-connect graphs where d_{max} is of order of n, the cost of this algorithm is higher than all the $O(mn)$ algorithms (although more markers are used than the $O(mn)$ algorithms).

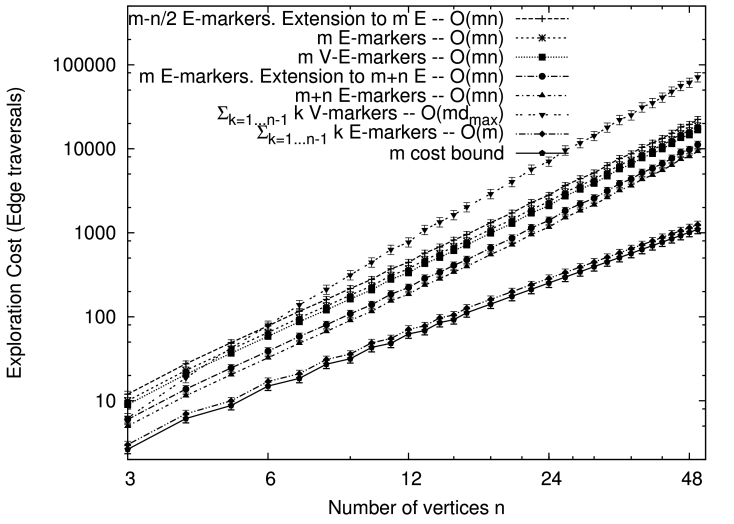
Also examined is the performance of the algorithm that extends the m E-marker algorithm, which requires no more than $m - \lfloor \frac{n}{2} \rfloor$ markers. In this extension, the sensed c at an explored

vertex no longer represents the exact number of explored edges at the vertex (but a subset of the explored edges). This information loss may potentially generate more potentially confusing verteces of the explored vertex. Results show that in some environments this generates a slightly increase in the cost, but the increase is minor (no more than 10%). Similar results are observed for the algorithm extending the $m + n$ E-marker algorithm, which requires no more than m markers. We also examined the marker consumptions of the two extensions. Actual number of markers used by the two extensions on lattice graphs is shown in Figure 6.12(a) and Figure 6.12(b) respectively. The maximum marker requirements of the two extensions ($m - \lfloor \frac{n}{2} \rfloor$ and m respectively), and the amount of markers used in the corresponding original algorithms (m and $m + n$ respectively) are also plotted. The actual amount of marker used is lower than the maximum number of markers required in the extensions. Compared against the original algorithms, substantial reduction on marker usage is observed and the reduction increases as the graph sizes increases. In the experiment, more than 50% reductions on marker usage is generated (but the cost is similar to the original algorithms.) Similar results are observed in densely connected graphs.

The performance of $k > 3$ (arbitrary) undirected markers algorithms are also examined empirically on the same set of graphs used above. We first run the version-1 approach (Algorithm 6.4) with different fractions of uniquely marked vertices. Results on the two types of environments are shown in Figure 6.13. Costs for the single directional V-marker algorithm and the $\sum_{k=1}^{n-1} k$ V-marker algorithm (100% vertices are uniquely marked), which are considered the upper and lower performance bound of the version-1 marker algorithm, are also present. When few vertices are

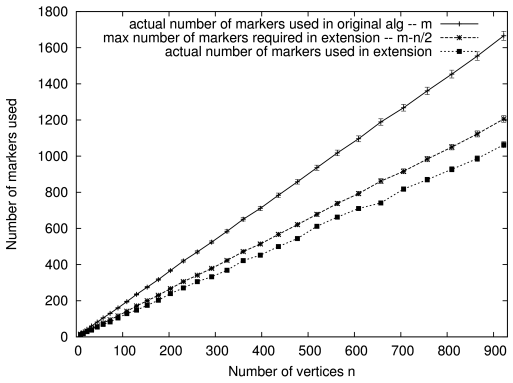


(a) Lattices with 10% edges removed

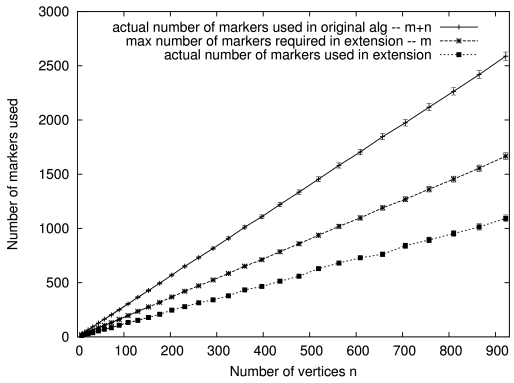


(b) Densely connected graphs with 10% edges removed

Figure 6.11: Performance of mapping with different class of multiple undirected marker aids (log scale). All results are averaged over 30 graphs. Each graph has randomly removed edges. Error bars show standard deviations.

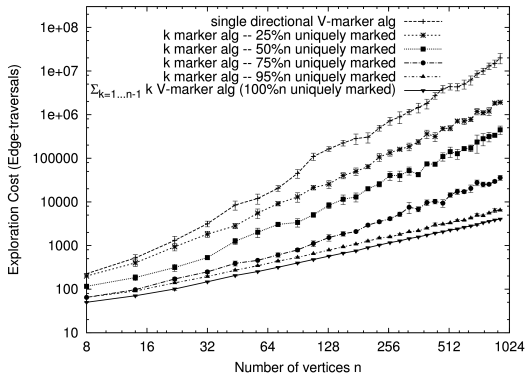


(a) $m - \lfloor \frac{n}{2} \rfloor$ E-marker algorithm. Extension to m E-marker algorithm.

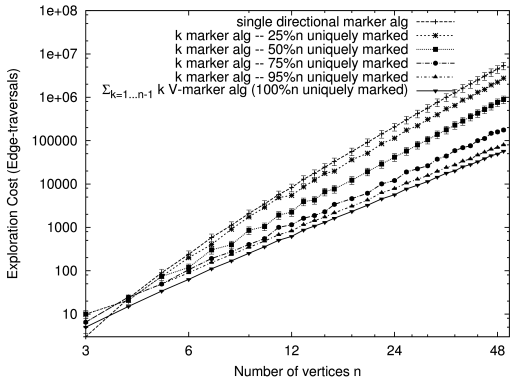


(b) m E-marker algorithm. Extension to $m + n$ E-marker algorithm.

Figure 6.12: Marker consumptions in the extension algorithms on lattice with 10% edges removed graphs of varying sizes. The upper bound of marker requirement in the extension algorithm and the number of markers used in their original algorithms are also plotted. Results are averaged over 30 graphs for each case. Error bars show standard deviations.



(a) Lattices with 10% edges removed



(b) Densely connected graphs with 10% edges removed

Figure 6.13: Performance of the version-1 k undirected marker algorithm (log scale). Mapping with different fractions of uniquely marked vertices. Results are averaged over 30 graphs. Each graph has randomly deleted edges. Error bars show standard deviations.

Marker classes (undirected immovable)			Upper bounds (edge traversals)
Number	**Drop Place**	**Constraints**	
1	–	–	N/A
2	vertex + edge (V-E-marker)	none	$O(m^2n)$
3	vertex, edge, or vertex+edge	none	$O(m^2n)$
unknown $k > 3$	vertex, edge, or vertex+edge	none	$O(m^2n)$
$m - \lfloor \frac{n}{2} \rfloor$	edge (E-marker)	closest-first	$O(mn)$
m	edge (E-marker)	closest-first	$O(mn)$
	edge (E-marker)	none	$O(mn)$
	vertex + edge (V-E-marker)	none	$O(mn)$
$m + n$	edge (E-marker)	none	$O(mn)$
$\sum_{k=1}^{n-1} k$	vertex (V-marker)	none	$O(md_{max})$
	edge (E-marker)	none	$O(m)$

Table 6.1: Solvability and known cost bounds of topological mapping with different undirected markers. The lower bound for the topological mapping problem is $\Omega(m)$ where $m = |E|$.

uniquely marked, the algorithm performance is close to the single directional marker algorithm. As the number of uniquely marked vertices increases, the exploration cost decreases. When most of the vertices are marked, the algorithm performance is close to the $\sum_{k=1}^{n-1} k$ V-marker algorithm. Similar results were observed when we run version-2 (Algorithm 6.5) with different fractions of (homogeneously) marked vertices. In that case, the costs of the single directional marker and the m V-E-marker algorithm can be considered the upper and lower performance bounds.

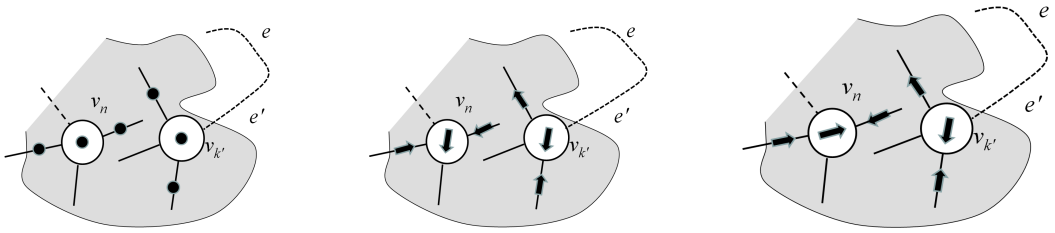
6.2.2 Mapping with multiple *directional* immovable markers

The previous section investigated the relative powers of different classes of multiple undirected immovable markers. The performances of these algorithms are summarized in Table 6.1. This section considers different classes of *directional* markers. If the homogeneous markers are direc-

tional, then the above multiple marker algorithms can be enhanced so their algorithmic costs are reduced, or the number of markers required are reduced.

Enhancements to the $O(mn)$ undirected marker algorithms

Above we have presented several algorithms that conduct search-based validation, and thus have $O(mn)$ cost bounds. These include two algorithms that use m undirected markers, and the extension algorithm of the E-marker algorithm which requires a maximum of $m - \lfloor \frac{n}{2} \rfloor$ markers. These also include the $m+n$ E-marker algorithm and the extension algorithm of it which requires a maximum of m markers. These algorithms can be improved with directional markers so cost is reduced, although the cost bound remains unchanged. Assume that when the robot drops a V-marker at a vertex, it drops the marker in such a way that the marker head points toward one of the exits (edges), and that when it drops an E-marker on an edge, it drops the marker in such a way that the marker head points toward one of the end vertices, as shown in Figure 6.14. Given this, as an example, the m undirected V-E-maker algorithm (Case 1b), which drops markers at all vertices and on all loop edges, can be improved as follows. Upon entering unknown end v_u via an edge e, if v_u and it edges contain markers, then in addition to the number of marked edges c which is used in the original algorithms, the robot extracts additional information that can be exploited. For example, based on the direction of the V-marker at $v_{k'}$, the robot can generate an ordered tuple *edgeINFO* that represents the presence (P) or absence (A) of E-markers on the incident edges at the vertex, enumerated starting from the edge pointed to by the directional V-marker. In the example in Figure 6.14(b), based on the direction of the V-marker,



(a) v_n is potentially confusing to $v_{k'}$.

(b) *edgeINFO* (P, A, P, A) vs. (A, P, A, P).

(c) *edgeINFO* (P_H, A, P_T, A) vs. (P_H, A, P_H, A).

Figure 6.14: Mapping with m directional V-E-markers. In (a) $v_{k'}$ and v_n are potentially confusing. In (b) the perception (signature) at $v_{k'}$ would be [4,V-1,E-1-H, E-3-T], generating two different *edgeINFO*. v_n is not potentially confusing. In (c) v_n is not potentially confusing.

the *edgeINFO* at $v_{k'}$ is (P, A, P, A). Based on the *edgeINFO* and other signature information, a known vertex in S is potentially confusing with $v_{k'}$ only if it satisfies two more conditions than that in the undirected version of algorithm: 1) the 1st edge on the *right* of the edge pointed by the V-marker, which which corresponds to the entry edge, must be an unexplored edge, 2) it has *edgeINFO* (P, A, P, A). Any vertex that does not satisfy all of these conditions (e.g., a vertex having *edgeINFO* (A, P, A, P)) is not potentially confusing and thus does not need to be visited. We can further improve the algorithms by considering the direction of the E-markers on the edges, i.e., incoming (P_H) or outgoing (P_T). Then more vertices in S might be disambiguated against $v_{k'}$. In the example, the *edgeINFO* would be enriched to (P_T, A, P_H, A), and vertices having different *edgeINFO* such as (P_H, A, P_T, A) are further disambiguated (Figure 6.14(c)). The other $O(mn)$ algorithms can be improved in a similar manner.

Enhancements to the $O(md_{max})$ undirected marker algorithm

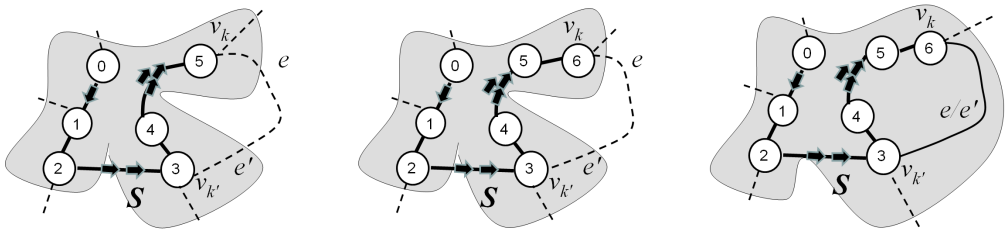
We have shown that with $\sum_{k=1}^{n-1} k$ undirected V-markers that are dropped at vertices, $O(md_{max})$ cost bound can be achieved (Algorithm 6.2). This cost is very high in densely connected graphs where d_{max} is large. If the V-markers are directional, then $O(m)$ cost bound can be achieved. Suppose that upon entering an unmarked vertex v_u, the robot drops the markers in such a way that the entry edge is indicated (all the markers dropped at v_u have the same direction). Clearly such an aid not only uniquely identify a visited vertex but it also defines an absolute ordering on each visited vertex, providing both explicit place and back-link information. That is, each visited vertex is a directional lighthouse.

We have also shown that with the same amount of undirected markers that are dropped at all non-loop edges, $O(m)$ cost bound can be achieved (Case 3b). With directional markers, while the cost bound can not be further reduced as it is already optimal, we can potentially reduce the number of markers that are used. In particular, we show below that the earlier idea of saving some E-markers on non-loop edges, which does not work for the case of undirected markers, can be applied to this algorithm in directional marker case. This is discussed next.

Enhancements to the $O(m)$ undirected marker algorithm

The $O(m)$ undirected marker algorithm (Case 3b) drops markers on all non-loop edges. Loop edges are not marked. Now with diectional markers, the key observation is that, unlike in the undirected E-marker case where we may need two marked edges to identify a vertex, here one (directed) marked edge is sufficient to identify a vertex as well as its edges. Given this, we try

to adopt the extension idea in Algorithm 6.3 that the robot only drops markers on *some* of the non-loop edges. Specifically, the robot explores in a closest-first pattern, and drops a unique number of markers on an edge $e = (v_k, v_u)$ only if e is a non-loop edge and its known end v_k has no marked edge yet. The robot does not drop markers on non-loop edge e if v_k has marked edge already, nor does it drop markers on loop edges. This extension idea works correctly when no loop is encountered: by enforcing a closest-first exploration and dropping markers on non-loop edges when v_k has no marked edge, each visited vertex is guaranteed to have at least one marked edge. The simple extension also works correctly if a loop is encounted and v_k has marked edge. These two working situations are shown in Figure 6.15(a). Unfortunately, this extension idea does *not* work when a loop edge e is explored (which is never marked) and v_k has no marked edges yet (Figure 6.15(b)). In this case, after exploring from v_k to $v_{k'}$, v_k still does not have marked edge yet, but the closest-first exploration leads the robot to continue by exploring unexplored edge at $v_{k'}$, leaving v_k unmarked, as shown in Figure 6.15(c). We need to ensure that whenever the robot explores to a visited place, the place has at least one marked edge. A simple approach to fixing this problem is to mark this kind of loop edges as well. That is, a loop edge is also marked if the known end v_k has no marked edge. In this approach robot drops markers if v_k has no marked edge yet. In the worst case, half of the edges are marked, and all the loop edges in the world are marked. So a maximum of $\sum_{k=1}^{\lceil \frac{m}{2} \rceil} k$ E-markers are required. Depending on m and n in the environment, this extension may require fewer markers than the original algorithm where $\sum_{k=1}^{n-1} k$ markers are used. For example, sparsely connected environments where m and n are close. Below we describe anther approach that guarantees the reduced use of markers.



(a) Exploring non-loop edges and loop edges where v_k has marked edge.

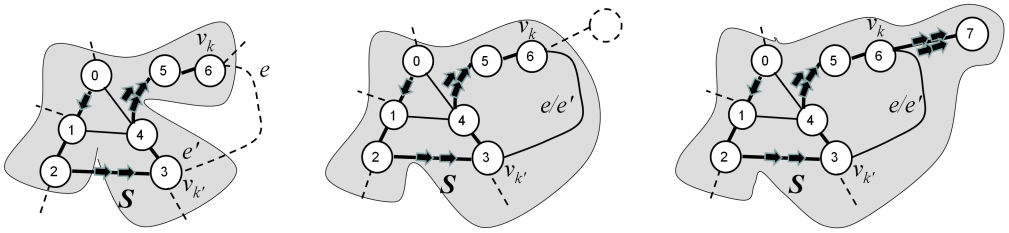
(b) Exploring an loop edge where v_k has no marked edge.

(c) Loop augmentation.

Figure 6.15: Trying to extend $\sum_{k=1}^{n-1} k$ E-marker algorithm by reducing markers on non-loop edges. Works in (1). Does not work in (2). In (2) exploring a loop edge where v_k has no marked edge. In (3) Next round explores from $v_{k'}$, leaving v_k unmarked.

In this alternative approach no extra markers are required. Intead of spending additional markers on loop edges, we require that, after exploring a loop edge whose known end v_k has no marked edge, rather than continuing from $v_{k'}$, the robot comes back to v_k and continue exploration there, unless v_k has no unexplored edges now. That is, if v_k has unexplored edge(s), the robot goes back to v_k and continues exploring other unexplored edges in v_k. If the unexplored edge explored in the next round is also a loop edge, the robot comes back again and continues from v_k again. The process repeats until either a non-loop edge at v_k is explored, or, v_k has no more unexplored edges. In the former case v_k will get a marked edge, as shown in Figure 6.16. In the later case the v_k remains unmarked but it is fully explored and is not accessible any more. In both the cases the robot resumes the normal closest-first exploration. With this further modification, the algorithm maintains true that each (accessible) visited place has at least one marked edge. Specifically, the following rules are enforced:

(1) select $e = (v_k, v_u)$ from U where v_k is the *closest* vertex to the current vertex.



(a) v_k is unmarked. e is a loop edge.

(b) Loop augmentation.

(c) Exploring an non-loop edge at v_k.

Figure 6.16: Mapping with $\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$ directional E-markers. After exploring e, v_k still does not have marked edges. Next round explores other edge at v_k. In (c) an non-loop edge of v_k is explored and marked.

(2) if e is an non-loop edge, mark e only if v_k has no marked edges.

(3) after exploring e, if e is a loop edge (so not marked) and v_k had no marked edge, then the robot return to v_k – next round will chose another unexplored edge at v_k.

The algorithm is given in Algorithm 6.6. The robot drops markers at non-loop edges only, and never on loop edges. In the worst case half of the non-loop edges are marked. So the maximum number of edges marked are $\lceil \frac{n-1}{2} \rceil$. So with directional markers, the number of markers required is reduced from $\sum_{k=1}^{n-1} k$ in the undirected marker algorithm to a maximum of $\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$.

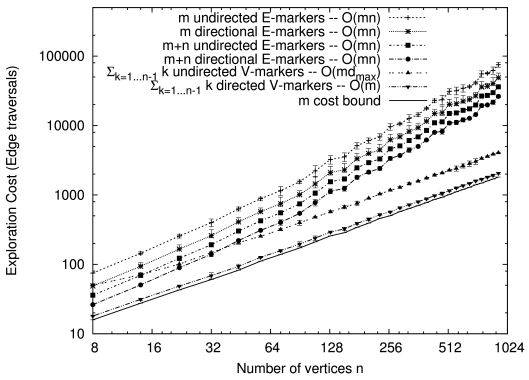
Empirical evaluations of mapping with multiple directional immovable markers

Experiments were conducted to examine the directional marker version of the search-based algorithms, in which *edgeINFO* can be derived and exploited. These directional marker version of algorithms still have $O(mn)$ cost bound but the real costs are expected to be reduced. Also examined is the $\sum_{k=1}^{n-1} k$ directional V-marker algorithm, which should have $O(m)$ cost bound

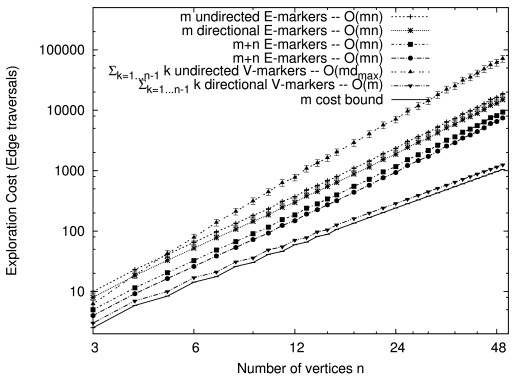
Algorithm 6.6: Mapping with $\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$ directional markers

```
1  S ← v_0; U ← edges in v_0;
2  i ← 1;
3  while U is not empty do
4      remove an unexplored edge e = (v_k, v_u) from U where v_k is closest;
5      the robot senses the marked edge(s) at v_k;
6      the robot traverses S to v_k and follows e to v_u;
7      the robot senses the marked edge(s) at v_u;
8      if v_u does not contain marked edge(s) then
9          if v_k has no marked edge(s) then
10             the robot goes back on e, drops i markers on e and back;
11             i ← i + 1;
               // if v_k has marked edges, leave e blank;
12         do 'non-loop augmentation' on S with v_u and e (with proper marker info on e);
13     else // v_u has marked edge(s)
14         identify v_u from S based on the unique markers at edge(s) v_u, say, v_k';
15         identify the label of e' based on its relative ordering to the known edges;
16         do 'loop-augmentation' on S with e/e' (with proper marker info on e);
17         remove e' from U;
18         if v_k has no marked edge(s) then
19             the robot comes back to v_k; // next round picks another edge in v_k;
20 return S;
```

now (improving on the $O(md_{max})$ cost bound of the undirected version of the algorithm). The two extension algorithms to the $\sum_{k=1}^{n-1} k$ undirected E-marker algorithm are also examined. With a reduced number of markers, the two extensions should still work correctly and maintain $O(m)$ cost bound. The algorithms are run on the same sets of graphs used earlier, and results are shown in Figure 6.17, where the costs of the corresponding undirected version algorithms and the optimal cost m are also plotted. The results confirm our analysis. The directional marker version of the m E-marker algorithm has lower cost than the undirected version algorithm. Similar cost reduction is observed for the directional marker version of the $m + n$ E-marker algorithm which



(a) Lattices with 10% edges removed

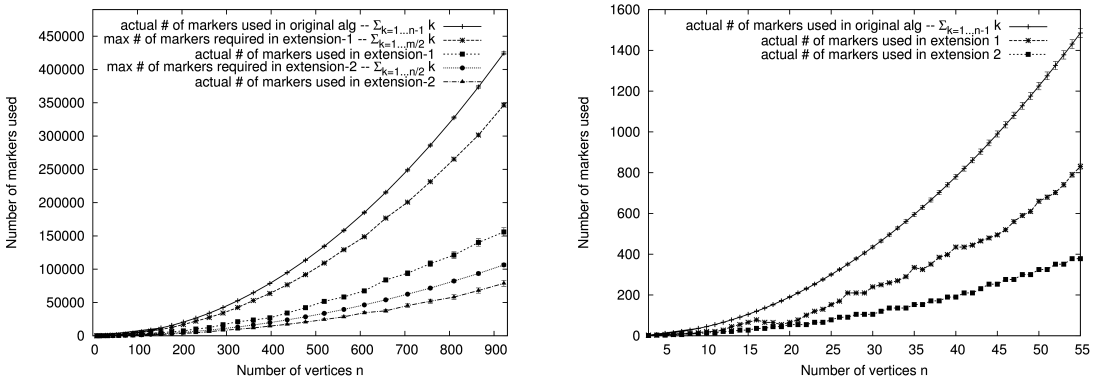


(b) Densely connected graphs with 10% edges removed.

Figure 6.17: Performance of several directional multiple marker algorithms. Results are averaged over 30 graphs. Each graph has randomly removed edges. Error bars show standard deviations.

has lower cost than the undirected version of $m+n$ E-marker algorithm. The directed version of the $\sum_{k=1}^{n-1} k$ V-marker algorithm now has cost near the optimal m cost. This is a reduction from the cost of the undirected version which has $O(md_{max})$ cost bound. The reduction is substantial in densely connected graphs where the undirected version algorithm has very high cost.

The two extension algorithms both work correctly and have costs that are very similar to the undirected version. The number of markers used in the two extension algorithms are also examined. Results for the extensions on the lattice graphs are shown in Figure 6.18(a), together with the marker usage in the original undirected marker algorithm ($\sum_{k=1}^{n-1} k$), and the maximum number of marker requirements in the extensions ($\sum_{k=1}^{\lceil \frac{m}{2} \rceil} k$ for extension-1 and $\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$ for extension-2). Results show that the actual marker used is consistently below the maximum requirements. Extension-2 requires lower number of markers than extension-1. The usage of markers on densely connected graphs is shown in Figure 6.18(b). It is interesting to observe that



(a) Marker used on lattice graphs

(b) Markers used on densely connected graph.

Figure 6.18: Markers used in the two extension algorithms. Each graph has randomly removed edges. Error bars show standard deviations.

although the maximum requirement of extension-1 is very high for the densely graphs (much higher than that the original algorithm), the actual marker usage is lower than the original algorithm. Similar to the case of lattice graphs, extension-2 has lower marker usage than extension-1.

6.3 Thread-based markers

V-markers are point markers and E-markers are short line segment markers. As summarized in Table 6.2, E-markers can be a more powerful marking aid, and so an interesting question becomes what happens if even stronger markers are considered. Consider the case of threads. A thread can be a potentially powerful marking aid in exploring topological worlds in that there are many ways of manipulating it, resulting in aids of different powers. The following sections examine different thread classes and for each identify their relative expressive power. Thread classes investigated in this section include very short threads, long threads, infinitely long threads and

Marker classes			Upper bounds	
Number	**Drop Place**	**Constraints**	**undirected**	**directional**
1	vertex or edge	none	–	$O(m^2n)$
2	vertex or edge	none	–	$O(m^2n)$
	vertex + edge (V-E-marker)	none	$O(m^2n)$	$O(m^2n)$
3	vertex or edge or vertex+edge	none	$O(m^2n)$	$O(m^2n)$
unknown $k > 3$	vertex or edge or vertex+edge	none	$O(m^2n)$	$O(m^2n)$
$m - \lfloor \frac{n}{2} \rfloor$	edge (E-marker)	closest-first	$O(mn)$	$O(mn)$
m	edge (E-marker)	closest-first	$O(mn)$	$O(mn)$
	edge (E-marker)	none	$O(mn)$	$O(mn)$
	vertex + edge (V-E-marker)	none	$O(mn)$	$O(mn)$
$\sum_{k=1}^{\lceil \frac{m}{2} \rceil} k$	edge (E-marker)	closest-first	$O(mn)$*	$O(m)$
$\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$	edge (E-marker)	closest-first + 'back to v_k'	$O(mn)$*	$O(m)$
$m + n$	edge (E-marker)	none	$O(mn)$	$O(mn)$
$\sum_{k=1}^{n-1} k$	vertex (V-marker)	none	$O(md_{max})$	$O(m)$
	edge (E-marker)	none	$O(m)$	$O(m)$

Table 6.2: Solvability and known cost bounds of topological mapping with different type and number of immovable markers. The trivial lower bound for the topological mapping problem is $\Omega(m)$ where $m = |E|$. * Note that $\sum_{k=1}^{\lceil \frac{m}{2} \rceil} k$ and $\sum_{k=1}^{\lceil \frac{n-1}{2} \rceil} k$ markers are used in directional marker case. In the case of undirected markers, no particular algorithm can be used, so the markers are used as in the the m marker case, which has $O(mn)$ cost.

the like[17]. Some thread classes can be mapped onto the various marker classes discussed above.

Assume that the robot is equipped with a thread, and that the robot can manipulate the thread in various ways. For example, it can 'tie' one end of the thread at a vertex and play that thread out as it moves through the graph, and perceive the thread when encountering the thread at a given vertex. The complete set of the robot's potential thread operations and perception are discussed below.

[17]The results have been published in [63]

6.3.1 Mapping with a very short thread

First consider the simplest threads. We present two algorithms for mapping deterministically a graph-like world with the shortest possible thread. In its simplest form the thread itself is unmarked (i.e., the surface of the thread provides no specific information), and is only long enough to be tied at a particular location (vertex) and then laid out in some direction. We call such a short thread a $l = \epsilon$ thread where l denotes the length of the thread. Such a thread can be considered as a generalization of a directional marker.

Mapping by fixing a $l = \epsilon$ thread (directional immovable marker)

Probably the simplest way to manipulate such a short thread is to tie it at the starting location, and never pick it up again. Suppose that the robot ties one end of the thread at its starting vertex v_0 and lays the thread out in the direction of one of the exits (edges) at v_0. This defines a unique directional marker at v_0. Assume the thread-related sensory information that the robot acquires at a vertex includes whether the thread is present at the vertex and the direction of the thread if the thread is present. Now the robot can apply the single directional immovable marker algorithm presented in Algorithm 3.1 or any of the enhancements to the algorithm, where the thread marked place is the lighthouse vertex v_0, and the entry edge can be examined by enumerating the exits (edges) and identifying the one that is pointed by the thread's free end.

Mapping by carrying a $l = \epsilon$ thread (directional movable marker)

The above approach has high exploration cost bound $O(m^2n)$, which can be reduced by increasing the power of the thread being used. One example is that the robot is not limited to only tying the thread once, but rather can also pick up and carry the thread, and tie the thread again at different vertices as desired. Assume that the robot can perform these operations. Assume the same thread-related perception as above.

The thread now can be used as a movable directional marker. A unique direction marker in a vertex provides both explicit place and back-link information. As mentioned in [27], we can run the single movable marker algorithm described in [26] but with reduced validation efforts. At each new place v_u via e, the robot ties the thread at v_u and lays it out toward the entry edge. The robot then searches S looking for the thread. If the thread is not found, then v_u has not been visited before. If the thread is found tied at some vertex $v_{k'}$ in S then both the place and back-link can be determined immediately due to the 'directional' thread in the vertex. The cost of the movable $l = \epsilon$ thread algorithm is still $O(mn)$ as the robot may have to visit each vertex in S in order to validate an edge (and its unknown end).

6.3.2 Mapping with longer threads

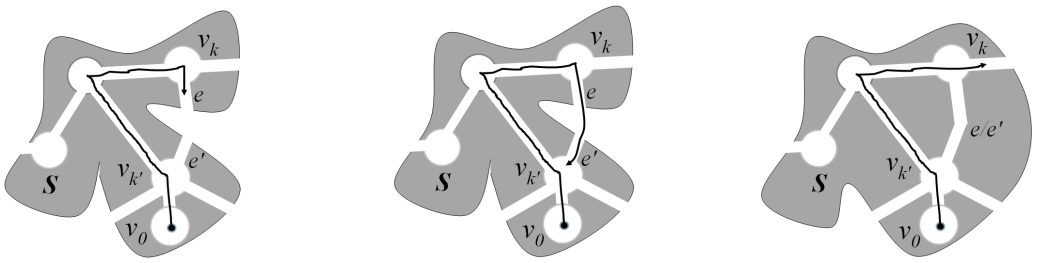
Threads of longer lengths can be used as a $l = \epsilon$ thread as described above, incurring $O(m^2n)$ or $O(mn)$ cost, but are there efficiencies to be found in using longer threads? The power of threads of various (known) lengths relative to the size of the world being explored is examined below.

Mapping with a $l = c(G)$ thread

First consider the case where the robot is assured that once the thread is tied, it can play the thread out and traverse to any other vertex without running out of thread, provided that the thread never forms a complete loop (cycle) during traversals. Model the problem as mapping with a thread of $l = c(G)$, where $c(G)$ denotes the *circumference*[18] of the graph-like world G.

Can the robot use such a thread to mark visited places so that exploration is more efficient than that in the $l = \epsilon$ thread case? The robot ties one end of the thread at the initial vertex v_0 and plays the thread out as it explores, marking each newly visited place (and up to two of its incident edges) by the thread, as shown in Figure 6.19(a). We enforce that the robot conducts the *closest-first* exploration, until it enters a marked place, or the current place become fully explored (i.e., has no more unexplored edges). In these cases the robot retraces its steps along the thread by picking up and rewinding the thread and tries to explore unexplored edges of other visited vertices along the thread. By consequence, the closest-first exploration ensures that each visited vertex that is not fully explored yet has been marked with the thread. Enforcing that the robot rewinds the thread after it enters (and validates) a thread-marked vertex (i.e., after exploring a loop edge) ensures that the robot never runs out of thread. The algorithm is outlined in Algorithm 6.7. Each step involves laying the thread along an unexplored edge e. At other end v_u the robot leaves the thread (free end) there (Figure 6.19(a)–(b)). The algorithm distinguish two possibilities at v_u: 1) v_u does not contain the thread and threaded edges before, 2) v_u (and

[18]The circumference of a graph is defined in graph theory as the length (number of edges) of the longest simple cycle in the graph (assuming a unit edge length).



(a) Closest-first exploration at unmarked (unvisited) places

(b) $v_u/v_{k'}$ contains thread

(c) Augments S by e/e', and rewinds thread

Figure 6.19: Mapping with a $l = c(G)$ thread. The robot rewinds the thread after exploring a loop edge, as shown in (b)–(c).

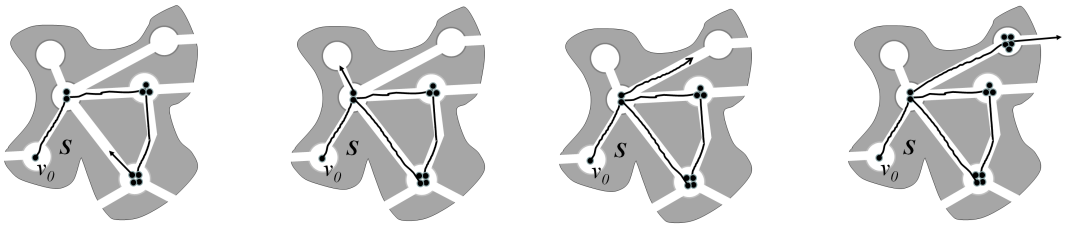
up to two of its edges) already contains the thread before. In case (1), v_u must have not been visited before. If v_u is fully explored now (i.e., has no more unexplored edges), the robot rewinds the thread along e back to v_k and tries unexplored edges there. Otherwise the robot continues with an unexplored edge at v_u. In case (2), further place validation and back-link validation may be needed. Similar to the footprints model algorithms, the newly laid thread (along e) is exploited. As one of the simplest approaches, validations are conducted by the robot (reversely) visiting vertices along the thread looking for the one that has one more thread-marked edge than it should have (based on S). While the approach has $O(mn)$ exploration cost as the robot may exhaust all the vertices currently on the thread (bounded by n) for validating a single edge, it is expected to produce a reduced cost over the short movable $l = \epsilon$ thread algorithms due to the reduced need for validation.

Algorithm 6.7: Mapping with a $l = c(G)$ thread

```
1  the robot ties the thread at v_0;
2  S ← v_0; U ← edges in v_0;
3  while U is not empty do
4      remove a closest edge e = (v_k, v_u) from U;
5      robot follows e to v_u, unwinding the thread along e to v_u;
6      if v_u does not contain the thread and threaded edges then
7          do 'non-loop augmentation' on S and U;
8          if v_u becomes fully explored now then
9              the robot rewinds thread to an vertex that is not fully explored;
10     else // v_u contains the thread already
11         the robot searches back along the thread looking for the vertex that has one more
           marked edge;
12         do 'loop augmentation' on S; remove entry edge e' from U;
13         the robot rewinds thread to a vertex that is not fully explored;
14 return S;
```

Mapping with a $l \gg m$ thread

We show here that if the robot is assured that the thread is much longer than the graph size (denoted as $l \gg m$), then the robot can map the world with $O(m)$ cost. With a $l \gg m$ thread, which guarantees that the robot does not run out of thread during exploration, the robot plays out the thread as it explores, even when the thread forms a loop. With such a long thread, there are several ways to alleviate validation efforts in each step. One approach here is to allow the robot to tie knots at each unknown place v_u it visits (Figure 6.20). The thread at a vertex v_u answers not only 'whether v_u has been visited before', but also 'exactly which vertex does v_u refer to'. Thus place validation for v_u is never needed, but when entering a thread-marked (visited) vertex $v_{k'}$ the robot may still need to conduct back-link validation. Similar to the $\sum_{k=1}^{n-1} k$ undirected V-marker approach, the robot exploits the existence of one of the known



(a) The closest-first exploration

(b) Do not need to rewind at a marked place

(c) Rewinds only when current place becomes saturated

(d) Continue exploration

Figure 6.20: Mapping with $l \gg m$ thread. Explore in a closest-first manner. Play the thread out as it explores. Tie different number of knots when entering a new place the first time. Each visited place is marked, and has at least two mared edges (except v_0). Rewinds the thread only when the current place is fully explored (b)-(c).

neighbors of $v_{k'}$ to identify the entry edge. Unlike in the $\sum_{k=1}^{n-1} k$ marker approach where the robot has to traverse each edge until it arrives at a known neighbor (thus have $O(md_{max})$ cost), here the robot can just traverse any one of thread marked edges to the other end vertex, which must be a known neighbor that is identified by its unique knot count. The exploration cost of the approach is $O(m)$.

6.3.3 Exploring with $l < c(G)$ (an arbitrary length) thread

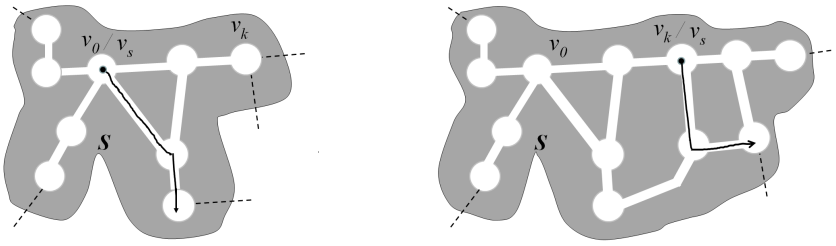
Now consider the general case that the the robot does not know *a priori* the relative length of the thread (relative to the environment size). The robot may run out of thread during exploration, even if the thread never forms a cycle. Possible approaches must consider both the case where the robot possesses thread, and the case where the robot runs out of thread. One possible approach is that, as in the arbitrary k markers algorithm, the robot uses thread to do validations when it possesses thread (phase-I), and when it runs out of thread (phase-II), it applies the hypothesis-

based traversal strategy developed for the single immovable marker algorithm. This approaches has an $O(m^2n)$ cost bound. Assuming that the robot can pick up the thread and tie it again, we present below two approaches that improve on this so that $O(mn)$ cost bound is obtained.

The explore-explore algorithm

The first approach extends the $l = c(G)$ algorithm. Initially the robot ties the thread at the starting vertex v_0 (call such a vertex v_s) and plays the thread out as it explores (in a closest-first fashion). As in the $l = c(G)$ algorithm, the robot rewinds the thread when it explores to a thread-marked place or when the current place becomes fully explored. The robot also rewinds the thread from the current vertex when thread length l is reached. (Note that the current vertex, which is visited now, becomes unmarked, although it may have notbeen fully explored yet and is thus accessible.) Eventually all the vertices within distance (the number of edges) l from v_s are explored, and the robot cannot continue exploration without running out of the thread. An example is shown in Figure 6.21(a). To continue exploration, the robot rewinds back to v_s, unties the thread and then chooses one of the (frontier) known vertices that is not yet fully explored as the new v_s, tying the thread there and continuing the exploration from the new v_s. By retying the thread, the robot then explores all vertices within a distance l from the new v_s (distance $2l$ from the previous v_s), and stops when it cannot continue exploration without running out of the thread again (Figure 6.21(b)). The process repeats until the environment is fully explored.

The algorithm is sketched in Algorithm 6.8. At each step, if v_u contains the thread then it has been visited before. As in the $l = c(G)$ algorithm, the robot searches back along the thread



(a) v_0 as v_s (b) $v_{k'}$ as the new v_s

Figure 6.21: Exploring with $l < c(G)$ thread. The 'explore-explore' approach. The thread has a length $l = 2$. In (a), all vertices within distance l of v_0 are explored. In (b), all vertices within distance $l$of the new v_s (within distance $2l$ of v_0) are explored. There are some vertices that are visited but unmarked, although they are not fully explored yet thus are accessible.

looking for the known vertex having one more thread-marked edges. If v_u does not contain the thread, then unlike in the $l = c(G)$ algorithm where an unmarked v_u must have not been visited before, here v_u may or may not be a visited place. This is due to the fact that the robot rewinds the thread whenever l is reached, leaving some visited places unmarked, although such visited places may have not been fully explored yet (Figure 6.21). To validate v_u and e, the robot search vertices that are *not* along the thread, looking for the one having a thread-marked edge now. The $l < c(G)$ thread algorithm has $O(mn)$ cost. Note that the movable $l = \epsilon$ and $l = c(G)$ algorithms discussed earlier can be considered special cases of this general algorithm.

The explore-merge algorithm

In the above algorithm a new place that does not contain the thread may or may not be a visited place. Thus when exploring to a place that contains no thread, the robot disambiguates the place against *all* of the known vertices in S – from both the current exploration phase and previous phases – that are potentially confusing. An alternative approach is to explore in

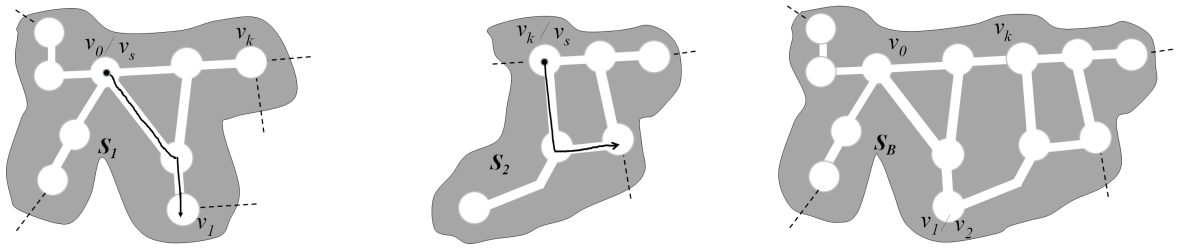
Algorithm 6.8: Mapping with a $l < c(G)$ thread. Explore-explore approach.

```
1  robot ties the thread at v_0 (call it v_s);
2  S ← v_0; U ← edges in v_0;
3  while U is not empty do
4      remove a closest edge e = (v_k, v_u) from U;
5      the robot traverses e to v_u; unwinds the thread to v_u;
6      if v_u contains the thread already then
7          the robot searches back along the thread looking for the vertex having one more
           stringed edge;
8          do 'loop augmentation' on S; remove entry edge e' from U;
9          robot rewinds the thread until it arrives at a not fully explored vertex;
10     else // v_u does not contain thread before -- may or may not a new place
11         robot searches non-thread vertices on S that has unexplored edges and has the
           same degree as v_u, looking for the vertex containing the thread's free end;
12         if the thread's free end is found at a vertex v_k' then
13             do 'loop augmentation' on S; remove entry edge e' from U;
14         else // thread not found, v_u is a new place
15             do 'non-loop augmentation' on S and U;
16     if thread length l is reached, or, the current place is fully explored now then
17         the robot rewinds thread looking for an not fully explored vertex along the thread;
18         if all the vertices along the thread (including v_s) are fully explored now then
               // all the vertices within distance l of v_s have been explored;
19             the robot traverses to v_s, unties the thread from v_s;
20             the robot chooses a not fully explored (frontier) known vertex as the new v_s,
               traverses there and reties the thread there (a new v_s);
21 return S;
```

'independent' phases. As in the above approach, the algorithm starts with an exploration phase in which the robot explores vertices within distance l of the starting vertex v_0, and then re-ties the thread at one of the known vertices v_s and explores again (Figure 6.22(a)). Unlike in the above approach, in this second exploration phase the robot disambiguates each unmarked place against known vertices generated in the *current* phase only, ignoring those known vertices explored in the previous phase. Specifically, instead of maintaining a consistent map S throughout the algorithm

execution, in the second phase the robot maintains a new map S_2 which initially contains only v_s where the thread is retied. During exploration the robot disambiguates new places only against known vertices within S_2 only. Note that in this case a vertex that is new in S_2 may be a visited place in the previous map S_1, and vice versa. To address this problem, when the second exploration phase finishes, a *merging* phase starts, in which the robot merges the two partial maps S_1 and S_2 using the merging algorithm described in [59, 61]. During merging, all vertices and edges in one map are disambiguated against those in the other map using some marking aid. One map is chosen as the *base map* S_B, which is augmented during merging. If a vertex or edge in the other corresponds to a vertex or edge in S_B, the vertex or edge is 'fused' (merged) into S_B (i.e., S_B is not augmented). Otherwise the vertex or edge is added into the base map as a new vertex or edge, augmenting S_B. During merging the thread could be used as a movable (directional) marker (See [59, 61] for details). The merging process finishes when all the vertices and edges in the two maps have been disambiguated. The merging result for the two partial maps in Figure 6.22(a) and Figure 6.22(b) is shown in Figure 6.22(c).

The robot then chooses one of the vertex in the merged map as the new v_s for the next exploration phase. From then on the algorithm proceeds by alternating phases of (independent) *exploration* and *merging* of the new partial maps to the base map S_B. In each merging phase that follows an exploration phase, partial map S_i from the preceding exploration phase is merged against the base map S_B, which is maintained and accumulated throughout the algorithm. The algorithm terminates when, at the end of a mering phase, S_B has no unexplored edges. Then the base map S_B is isomorphic to the world model G ([59, 61]). The algorithm is listed in



(a) The first exploration phase, v_0 as v_s

(b) The second exploration phase, v_k as v_s

(c) The merged result of the two maps

Figure 6.22: Exploring with $l < c(G)$ thread. The 'explore-merge' approach. Assume that the thread has a length $l = 2$. In (a), all vertices within distance two of v_0 are explored. In (b), vertex v_2, which corresponds to v_1 in S_1, is a new vertex in S_2. In (c), vertex v_2 in S_2 and vertex v_1 in S_1 are 'fused'.

Algorithm 6.9. Both the exploration and merging phases has $O(mn)$ cost bound. The algorithmic cost bound of this approach is thus $O(mn)$.

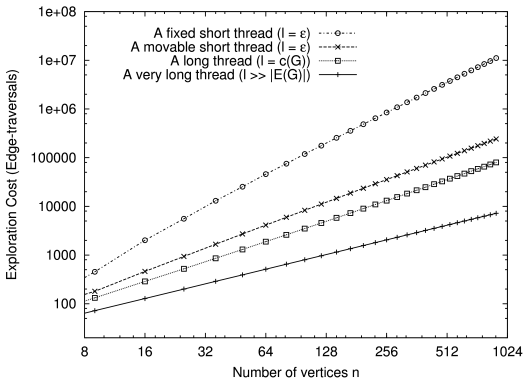
The potential cost reduction of this 'explore-merge' approach over the 'explore-explore' approach results from the fact that in each exploration phase disambiguations are constrained within the current subgraph S_i only, and the fact that in the merging process some vertices and edges can be disambiguated without traversals (see [59]). One of the potential drawbacks of this approach is that in different exploration phases the robot may explore edges and vertices that are already explored in previous exploration phases, due to the constrained disambiguations in each exploration phase. This cannot happen in the 'explore-explore' algorithm where disambiguations are conducted against *all* known places in the 'global' map S.

Algorithm 6.9: Mapping with $l < c(G)$ thread. Explore-merge approach. Alternating exploration and merging phases

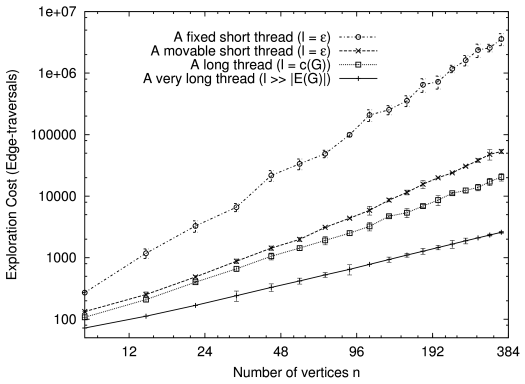
// start with two phases of independent exploration;
$S_1 \leftarrow v_0$; *// the map representation used in the next exploration phase*;
$v_s \leftarrow v_0$;
the robot ties the thread at v_s and explores all vertices within distance l of v_s, using S_1 as the map representation;
$v_s \leftarrow$ an unsaturated vertex in S_1;
$S_2 \leftarrow v_s$; *// the map representation used in the next exploration phase*;
the robot reties the thread at v_s and explore all vertices within distance l of v_s, using S_2 as the map representation;
// now merge the two partial maps S_1 *and* S_2;
$S_B \leftarrow S_1$; *// choose* S_1 *as the base map*;
merge S_2 into S_B; *//* S_B *is the merging result, getting augmented*;
while S_B *has unexplored edges* **do**
    // alternating phases of exploration and merging;
    $v_s \leftarrow$ an not fully explored vertex in S_B;
    $S_i \leftarrow v_s$; *// the map representation used in the next exploration phase*;
    the robot reties the thread at v_s and explores all vertices within distance l of v_s, using S_i as the map representation;
    merge S_i into S_B; *//* S_B *is the merging result, getting augmented*;
return S_B;

Empirical evaluations

Here we present empirical comparisons of the performance of mapping with the different thread algorithms discussed above. We first compare the relative power of a fixed short ($l = \epsilon$) thread, a movable short ($l = \epsilon$) thread, a long ($l = c(G)$) thread, and a very long thread ($l \gg |E(G)|$). Experimental results are reported for both homogeneous and non-homogeneous lattice graphs of varying sizes (see Figure 6.23). For these environments, exploring by carrying the short ($l = \epsilon$) thread which has $O(mn)$ cost, obtains a reduced exploration cost over exploring with a fixed short thread which has $O(m^2n)$ cost. Exploring with $l = c(G)$ thread, which also has $O(mn)$ cost bound, provides a further cost reduction over exploring with short threads. Exploring with



(a) Homogeneous lattices.



(b) Lattices with 10% edges removed.

Figure 6.23: Performance of mapping on lattices of varying sizes using different threads (log scale). Results for (b) are averaged over 30 graphs. Each graph has randomly deleted edges. Error bars in (b) show standard deviations.

a very long thread (with knots) which has a $O(m)$ cost bound, provides the lowest cost over all the thread classes considered.

The results for the $l < c(G)$ algorithm with varying thread length l are shown in Figure 6.24. The algorithm demonstrates cost reduction as l increases. Note that when the thread is sufficiently long ($l \geq c(G)$), fixed exploration cost is produced (the algorithm acts as the $l = c(G)$ algorithm).

6.4 Summary

This chapter investigates the relative power of markers on edges, multiple immovable markers and thread-based markers. Similar to the case of vertex markers, a undirected edge marker is not always sufficient for a robot to map an arbitrary world deterministically, although again similar to the vertex marker case, a directional edge marker is. For multiple immovable markers,

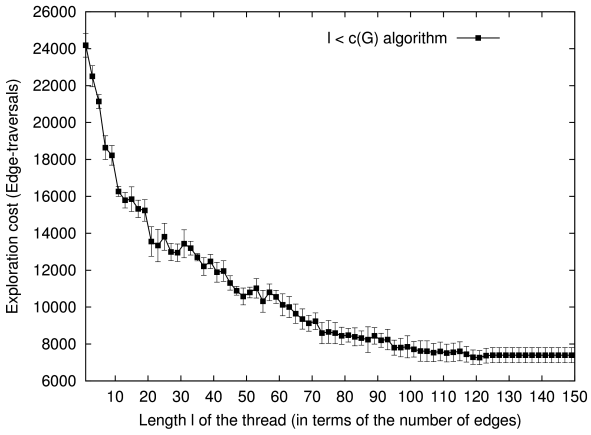


Figure 6.24: Performance of mapping on non-homogeneous lattice (20×20 vertices lattice with 10% edges removed) using strings of different lengths. Results are averaged over 30 graphs each has randomly removed edges. Error bars show standard deviations.

three different classes of multiple marker algorithms were developed: algorithms that only drop markers at vertices (V-marker algorithms), algorithms that only drop markers on edges (E-marker algorithms) and algorithms that drop markers both at vertices and on edges (V-E-marker algorithms). As summarized in Table 6.2 , as the number and nature of the marker is augmented more efficient algorithms emerge. This chapter also considers the power of thread-based markers. There are many ways of manipulating a thread, resulting in aids of different powers. Given the simplest form of thread, deterministic mapping is possible with cost $O(m^2n)$. The minimum cost for mapping is $O(m)$ and this can be obtained with a sufficiently long thread. Overall the longer the thread the lower the cost. Performance bounds of the various classes of threads discussed in this chapter are summarized in Table 6.3.

Thread classes		**Upper bounds**
Length of thread	**Need for retying**	
a very short thread $l = \epsilon$	no	$O(m^2n)$
	yes	$O(mn)$
an arbitrary length thread $l < c(G)$	yes	$O(mn)$
a long thread $l = c(G)$	no	$O(mn)$
a very long thread $l \gg m$	no	$O(m)$

Table 6.3: Solvability and cost bounds of different threads.

.

Chapter 7

Summary and Future work

7.1 Summary

Within a topological formalism, this thesis examines the relative expressive power of different marking aids in solving the SLAM problem deterministically. This thesis also explores non-deterministic approaches that map the world with insufficient supply of marking aids, including approaches that map the world without resorting to marking aids at all.

1. Given a topological world, it is not, in general, possible to map the world deterministically without resorting to the use of sufficient place and back-link information to solve the loop closing problem. But what information is sufficient? This thesis first shows that neither can explicit place information or explicit back-link information alone enable the robot to solve the problem. This thesis then shows that a deterministic solution is possible, however, if both explicit place and back-link information exist in one vertex. Specifically, this thesis demonstrates that given a single 'directional lighthouse' vertex that provides both such information locally, a provably correct mapping strategy using such information exists. Such information can be established in a number of ways including through the addition of

a single directional immovable marker to the environment, or through the use of multiple undirected immovable markers. In terms of the number of markers and the movability of markers, a single immovable marker is considered the minimum marker case.

- This thesis develops an algorithm with which, by exploiting the directional lighthouse established with a single directional vertex marker, the robot can determine its location and back-link at each visited place, mapping the world deterministically. Unlike the earlier movable marker algorithm [26] where place and back-link at each step are determined separately, in the algorithm developed here, both place and back-link validation are validated simultaneously. In particular, potential loop closing hypotheses are developed, each containing a possible location *and* back-link. Paths from the hypotheses to the lighthouse location (through the already explored world) and expected perceptions along the paths are also computed. The hypotheses are then validated by having the robot traverse each of these paths, comparing the observed perceptions against the expected perceptions. A loop closing hypothesis is confirmed only if the observed perceptions match the expected perceptions throughout path traversal, and is rejected otherwise. The 'new' location is a previously explored location if one of the hypotheses is confirmed, and is distinct from the explored locations if no hypothesis exists or all the hypotheses are rejected. This algorithm has a tight lower cost bound $\Omega(m)$, and an upper cost bound $O(m^2n)$ is derived. A correctness proof of the algorithm is presented. Earlier algorithms such as [26] required the robot to manipulate markers in order to map the world. With the development of the directional lighthouse

algorithm, it is now possible for the robot to map a topological world deterministically without manipulating markers.

- The algorithm is evaluated both via simulation and on a real robot systems. Most of the deterministic topological mapping algorithms have only been implemented in simulation and it has been an open question as to how well the assumptions of these algorithms would transfer to real robots. Evaluation of the algorithm on a real robot shows that the basic sensing and locomotion assumptions that underlie the algorithm are realistic when applied to real world environments, sensors and robotic platforms.
- This thesis explores two classes of enhancements to the basic directional lighthouse algorithm. The first class exploits executed traversals to reject potential hypotheses. These approaches try to reduce the number of hypotheses that requires motion, and the amount of motion required in validating the hypotheses. The second class of approaches requires extra traversals to construct expanded local signatures. With the expanded local signatures, the number of hypotheses of a newly explored edge can be potentially reduced.

2. This thesis considers mapping with less marker information.

   - This thesis considers mapping with an undirected immovable vertex marker. The key is that while a single undirected immovable marker is not sufficient in general, it can be used to map the world that contains some structure that, once marked, forms a unique signature which provides both explicit place and back-link information. Such a world

can be mapped deterministically with an undirected vertex marker, using the single directional marker algorithm or its enhancements. Since the single undirected vertex marker can be used to provide explicit place information at a vertex, this implies that as long as there exists a vertex in the world that can provide explicit back-link information, then this vertex can be used to establish the directional lighthouse, enabling the robot to map the world deterministically.

- This thesis considers mapping without any markers. Without any marker, we must rely on the structure of the environment. It examines exploiting local and global signatures during exploration. Both approaches are non-deterministic in that the algorithms may not generate any world model or may generate one or more world models but cannot determine which model is the true representation of the world.

3. This thesis investigates the relative power of markers on edges, multiple immovable markers and thread-based markers.

   - Similar to the case of vertex markers, an undirected edge marker is not sufficient for a robot to map an arbitrary world deterministically, but a directional edge marker is.
   - For multiple immovable markers, three different classes of multiple marker algorithms were developed: algorithms that only drop markers at vertices (V-marker algorithms), algorithms that only drop markers on edges (E-marker algorithms) and algorithms that drop markers both at vertices and on edges (V-E-marker algorithms). The major optimization results obtained with different collections of markers and structured

exploration patterns are summarized in Table 6.2.

- There are many ways of manipulating a thread, resulting in marker aids of different powers. Overall the longer the thread the lower the cost. Performance bounds of the various classes of threads discussed in this work are summarized in Table 6.3.

7.2 Future work

This thesis work suggests several directions that are worth exploring in the future. We observed that the derived upper cost bound $O(m^2n)$ of the basic single directional immovable marker algorithm is quite loose for some environments. On several environments such as the lattice hole graphs, the actual costs of the algorithm are much lower than the cost bound. In deriving the upper cost bound, for each newly explored edge e, we bound the number of hypotheses by the worst case scenario that all of the current unexplored edges incident on non-marked places are potential hypotheses, and we bound the length of each path by the number of all current nodes, and assume the worst case scenario that all the hypotheses are examined, during which each path traversal comes to the end of the path. Intuitively this is a very 'pessimistic' assumption that might never happen. Thus, it is worth investigating whether the worst case scenarios exist (at the same time). If these worst case scenarios can happen together for some environment, then what classes of graphs are they? If it can be justified that such worst case scenarios do not exist, then it would be an interesting direction for future work to investigate whether a tighter upper bound of the algorithm can be derived. In both cases, it would be interesting to derive tighter cost bounds for some fundamental environments such as the lattice hole graphs. Similarly, it would

be an interesting direction for future work to derive tighter cost bounds for the enhancements to the basic algorithm. In addition to m and n, it might be helpful to derive the bound by considering some other factors such as graph diameter and maxmium vertex degrees.

This thesis shows that as long as a 'directional lighthouse' can be established in the world, the world can be mapped deterministically. An interesting question here is that whether such directional lighthouse information is the minimum information required for a deterministic solution for topological SLAM. That is, whether mapping can be conducted deterministically with less information. (How to define 'less' is another interesting question). For example, can the world be mapped deterministically with a explicit place information and some ambiguous back-link information, or conversely, with a explicit back-link information and some ambiguous place information? Related question here is about the definition of 'minimum' markers, which is difficult to develop in the global sense. Given that a single movable marker is sufficient to solve the SLAM problem deterministically, in this thesis we consider the number of markers as the main dimension of marker complexity, and then for a given number of markers we consider the movability of the marker. Given this local definition, a single immovable marker, which does not involve robot operation on it, is considered the minimum marker. It is interesting to consider if there are other definitions of minimalist that are more realistic than the definition used in this thesis.

For the other marking aids investigated in this thesis, it is worth further investigating the power of thread-based markers. For example, in this thesis we considered threads that are in the simplest form where a thread contains no marks on its surface. It is interesting to consider more

complicated forms of threads such as those with painted marks on the surfaces. It must also be interesting to investigate how thread-based markers can be exploited in the metric framework. It is also interesting to investigate how the thread-based markers as well as the other marker classes discussed in this thesis can be used for mapping *directed* graphs, examining if mapping of directed graphs be tackled with the methods presented in the thesis.‘

For marker-less exploration, it is interesting to investigate the expressive power of the prior knowledge of the probabilistic distribution of some environmental properties. This information may include, as discussed earlier, the probability distribution of the diameter of the graph-like environment, the probabilities of different vertex degrees, the probability distribution on the number of vertices in the environment (e.g., number of vertices n is a Gaussian distribution function), and the like. Such information could be used to assign probabilities to different world models. It is an interesting future work to categorize different prior information about the probabilistic distribution and investigate the power of this information on algorithm performance.

Another interesting future work is to investigate how the ‘directional lighthouse’ information can be exploited in the traditional metric framework (such as the one described in Section 2.2). In particular, how a directional lighthouse information can be established, identified and exploited in presence of (noisy) metric data. One of the challenges facing metric SLAM approaches is the large number of hypotheses (world models) that need to be maintained during exploration. It is expected that the directional lighthouse information can be exploited to alleviate this challenge. For example, we can run a particle-filter based SLAM algorithm which takes as input laser sensor data and maintains a geometric world. Given the metric measurements, it is expected

that unique position and orientation information can be captured from the environment during exploration (we can create some directional landmarks otherwise). During exploration when the robot sees the unique lighthouse or a relatively rare (likely unique) location, it can collapse the hypotheses (world models) that don't think this is a lighthouse or a rare place. A rare place is not perfect (thus is not used for hypothesis validation in our deterministic approaches), but is a good evidence in the probabilistic framework. We can collapse the hypotheses by assigning high probabilities to the hypotheses that matches the observations and low probabilities to those that don't. Another situation where lighthouse information could be exploited is when solution space becomes unmanageable. Now we can examine a hypothesis by trying to drive the robot back to a directional lighthouse or a relative rare location, based on the hypothesized world model. Then if the robot senses the lighthouse or a rare location with the expected orientation, then this hypothesis is given a high probability and is given a very low probability otherwise. In the ideal case, using the directional lighthouse information would enable the loop closing to be solved deterministically, but this might not be true in practice, given the sensing errors. It is worth exploring how the technique is utilized in the presence of (noisy) metric measurements. I envision that this is an interesting future work, and is the practical application of the theoretic result given in this thesis.

Bibliography

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[2] M. O. Albertson and K. L. Collins. Symmetry breaking in graphs. *Electronic Journal of Combinatorics*, 3:1–8, 1996.

[3] F. Amigoni, S. Gasparini, and M. Gini. Map building without odometry information. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3753–3758, New Orleans, LA, USA, 2004.

[4] B. Awerbuch, M. Betke, R. L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152:155–172, 1999.

[5] T. Bailey and H. Durrant-Whyte. Simultaneous Localization and Mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine*, 13:108–117, 2006.

[6] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40:143–162, 2007.

[7] M. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: exploring and mapping directed graphs. *Information and Computation*, 176:1–21, 2002.

[8] M. Bender and D. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 75–85, Santa Fe, NM, USA, 1994.

[9] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18:231–254, 1995.

[10] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, Ann Arbor, MI, USA, 1978.

[11] M. Blum and W. J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 147–161, Providence, RI, USA, 1977.

[12] J. Chalopin, S. Das, and N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *International Symposium on Distributed Computing (DISC)*, pages 108–122, Lemesos, Cyprus, 2007.

[13] A. Chopra, M. Obsniuk, and M. Jenkin. The Nomad 200 and the Nomad SuperScout: Reverse engineered and resurrected. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 55–63, Quebec City, Quebec, 2006.

[14] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137, 2001.

[15] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385:34–48, 2007.

[16] A. De, J. Lee, and N. J. Cowan. Toward SLAM on graphs. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 631–645, Guanajuato, Mexico, 2008.

[17] X. Deng, E. Milios, and A. Mirzaian. Robot map verification of a graph world. *Combinatorial Optimization*, 5:383–395, 2001.

[18] X. Deng and A. Mirzaian. Robot mapping: Foot-prints versus tokens. In *Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC)*, pages 353–362, Hong Kong, 1993.

[19] X. Deng and A. Mirzaian. Competitive robot mapping with homogeneous markers. *IEEE Transactions on Robotics and Automation*, 12:532–542, 1996.

[20] X. Deng and C. Papadimitriou. Exploring an unknown graph. In *31st Annual Symposium on Foundations of Computer Science*, pages 355–361, St. Louis, MO, USA, 1990.

[21] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326:343–362, 2004.

[22] 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.

[23] G. Dudek, P. Freedman, and S. Hadjres. Mapping in unknown graph-like worlds. *Journal of Robotic Systems*, 13:539–559, 1998.

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

[25] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Using multiple markers in graph exploration. In *SPIE Vol. 1195 Mobile Robots IV*, pages 77–87, Philadelphia, USA, 1989.

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

[27] 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.

[28] 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.

[29] G. Dudek and D. Marinakis. Topological mapping with weak sensory data. In *AAAI Conference on Artificial Intelligence*, pages 1083–1088, Vancouver, BC, Canada, 2007.

[30] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. *ACM Transactions of Algorithms*, 2:380–402, 2006.

[31] H. Durrant-Whyte and T. Bailey. Simultaneous Localization and Mapping (SLAM): Part I. *IEEE Robotics and Automation Magazine*, 13:99–100, 2006.

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

[33] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *13th Annual European Symposium on Algorithms (ESA)*, pages 11–22, 2005.

[34] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–10, Washington, DC, USA, 2009.

[35] 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.

[36] J. L. Gross and T. W. Tucker. *Topological Graph Theory.* Wiley-Interscience, New York, NY, USA, 1987.

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

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

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

[40] 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.

[41] B. Kuipers, D. Tecuci, and B. Stankiewicz. The skeleton in the cognitive map: A computational and empirical exploration. *Journal of Environment and Behavior*, 35:81–106, 2003.

[42] S. Kwek. On a simple depth-first search strategy for exploring unknown graphs. In *5th International Workshop on Algorithms and Data Structures (WADS)*, pages 345–353, London, UK, 1997. Springer-Verlag.

[43] D. Marinakis and G. Dudek. Pure topological mapping in mobile robotics. *IEEE Transactions on Robotics*, 26:1051–1064, 2010.

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

[45] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999.

[46] P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36:96–103, 2000.

[47] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, Kobe, Japan, 2009.

[48] M. Rabin. Maze threading automata. Presented at MIT and UC Berkley, 1967.

[49] A. Ranganathan and F. Dellaert. Inference in the space of topological maps: an MCMC-based approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1518–1523, Sendai, Japan, 2004.

[50] A. Ranganathan and F. Dellaert. Data driven MCMC for appearance-based topological mapping. In *Robotics: Science and Systems (RSS)*, pages 209–216, Cambridge, MA, USA, 2005.

[51] A. Ranganathan and F. Dellaert. A Rao-Blackwellized particle filter for topological mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 810–817, Orlando, FL, USA, 2006.

[52] A. Ranganathan, E. Menegatti, and F. Dellaert. Bayesian inference in the space of topological maps. *IEEE Transactions on Robotics*, 22:92–107, 2006.

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

[54] W. J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences (JCSS)*, 7:389–403, 1973.

[55] S. Thrun. Robotic mapping: a survey. In *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] T. W. Tucker. Distinguishing maps. *Electronic Journal of Combinatorics*, 18, 2011.

[58] S. Tully, G. Kantor, H. Choset, and F. Werner. A multi-hypothesis topological SLAM approach for loop closing on edge-ordered graphs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4943–4948, St. Louis, MO, USA, 2009.

[59] H. Wang. Multiple robot graph exploration. Technical Report CSE-2007-06, Department of Computer Science and Engineering, York University, 2007.

[60] H. Wang, M. Jenkin, and P. Dymond. Enhancing exploration in graph-like worlds. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 53–60, Windsor, ON, Canada, 2008.

[61] H. Wang, M. Jenkin, and P. Dymond. Graph exploration with robot swarms. *International Journal of Intelligent Computing and Cybernetics*, 2:818–845, 2009.

[62] H. Wang, M. Jenkin, and P. Dymond. It can be beneficial to be lazy when exploring graph-like worlds with multiple robots. In *The IASTED International Conference on Advances in Computer Science and Engineering (ACSE)*, pages 55–60, Phuket, Thailand, 2009.

[63] H. Wang, M. Jenkin, and P. Dymond. Using a string to map the world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 561–566, Taipei, Taiwan, 2010.

[64] H. Wang, M. Jenkin, and P. Dymond. The relative power of immovable markers in topological mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1050–1057, Shanghai, China, 2011.

[65] H. Wang, M. Jenkin, and P. Dymond. Enhancing topological exploration with multiple immovable markers. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 322–329, Toronto, ON, Canada, 2012.

[66] H. Wang, M. Jenkin, and P. Dymond. Enhancing exploration in topological worlds with a directioinal immovable marker. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 348–355, Regina, SK, Canada, 2013.

[67] H. Wang, M. Jenkin, and P. Dymond. Deterministic topological visual SLAM. In *The International Symposium on Information and Communication Technology (SoICT)*, Hanoi, Vietnam, 2014.

[68] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[69] F. Werner, C. Gretton, F. Maire, and J. Sitte. Induction of topological environment maps from sequences of visited places. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2890–2895, Nice, France, 2008.

Appendix A

Notation

Symbol	Meaning
G	An undirected embedded graph representing the graph-like world being explored.
m	The number of edges in the graph-like world G.
n	The number of vertices in the graph-like world G.
S	Partial map representation maintained during exploration, representing currently explored subgraph of the world.
U	Set of unexplored edges that emanate from vertices in S.
v_k	The known (visited) end vertex of an unexplored edge.
v_u	The unknown (unvisited) end vertex of an unexplored edge.
h	A loop closing hypothesis, which include an edge and its known end vertex.
H	Set of loop closing hypotheses.
$\mathcal{M}_{h'}$	A motion sequence for the hypothesis h'.
$\mathcal{P}_{h'}$	A perception sequence obtained in executing a motion sequence for h'.
$\mathcal{P}^E_{h'}$	Expected perceptions in executing a motion sequence for h'.
M	A world model maintained in marker-less exploration.
T	A tree of world models M, maintaiend in marker-less exploration.
TM	Traversal map used in marker-less exploration.

Appendix B

Glossary

Term	Symbol	Meaning
Back-link validation		When the robot enters an known place, the process of determining the entry edge at the place.
Directional lighthouse		A vertex that contains both explicit place information and explicit back-link information. Upon entering a directional lighthouse, the robot knows the identity of the (known) vertex it is visiting, and the entry edge by which it enters the vertex.
Execution path		The actual path (traversed or would be traversed) in executing a motion sequence.
Expanded signature		The signatures of a vertex that also includes the local signature of its neighborhoods.
Expected perceptions	$\mathcal{P}^E_{h'}$	Perception sequence that should be obtained if the hypothesis h' is true.
Explicit back-link information		Unambiguous information on the entry edge at the vertex that the robot is currently in.
Explicit place information		Unambiguous information on the identity of the vertex that the robot is currently in.
Loop augmentation		Augmentation of the partial map S when a loop edge is explored. The map is augmented with an edge.

Term	Symbol	Meaning
Loop closing hypothesis	h'	A hypothesis about the place the robot is currently in as well as the back-link (entry edge) throught which the robot entered the current place. A hypothesis includes an unexplored edge and its known end vertex.
Loop edge		A newly explored edge that leads the robot to an visited place.
Motion sequence	$\mathcal{M}$	The sequence of relative edge orderings with respect to the entry edges along which the robot enters each vertex.
Non-loop edge		A newly explored edge that leads the robot to a new place.
Non-Loop augmentation		Augmentation of the partial map S when a non-loop edge is explored. The map is augmented with an edge and a vertex.
Perception sequence	$\mathcal{P}$	The sequence of signatures of vertices visited during executing of a motion sequence.
Place validation		The process of determining if a newly visited place is truly distinct from all the previously visited places, or it corresponds to some known vertex.
Signature		The sensed distinctive property of a vertex, which consists of the degree of the vertex and the marker information on the vertex, including the presence or absence of the marker(s) at the vertex and other marker-related information (e.g., number directionality), denoted [deg, V#-dir, otherInfo]. Also referred to as **local signature**.
Traversal map	TM	The map which is non-loop augmented at every step. A traversal map is maintained in marker-less exploration and is used to conduct a Breadth-first search (BFS) on the environment.