Exploring Topological Environments

Hui Wang

Technical Report CSE-2010-05

June 2010

Department of Computer Science and Engineering
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

Abstract

This report reviews current research trends related to the problem of robotic exploration and mapping. In exploring and mapping an unknown environment, one fundamental problem is answering the question 'have I been here before?' (This is also known as the 'loop closing' problem.) Answering this question involves disambiguating the current place of the robot against previously visited or known locations. Two fundamental approaches to solving the problem are reviewed. The first approach resorts to the use of an 'oracle' to help solve the disambiguation problem. Oracles are available with different disambiguating powers, and the relative strengths of different oracles are explored. The second approach exploits augmenting locations with metric information that describes the underlying environment. Open questions are discussed.

Table of Contents

1 Introduction

Acquiring a map is a fundamental task in robotics. 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 safe and simpler environments such as the interiors of buildings, accurate mapping of the environment is important. Without a map, many robotic tasks become difficult or even impossible. In addition to serving as the basis for robotic motion planning, accurate maps also have practical significance. For example, a mine accident trapped nine miners for nearly four days after they accidentally drilled into a nearby abandoned mine due to the use of inaccurate maps [58]. (This accident has motivated extensive investigation into the deployment of mobile robots and more advanced approaches as a possible technology for acquiring accurate maps of abandoned mines and other dangerous places [78].)

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 itself within the map as it 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.

A critical issue for SLAM is how to model the underlying environment. One fundamental representation is based on a topological or graph-like formalism which models the environment as a graph. A graph-like world represents the minimal information that a robot must be able to represent in order to distinguish one place from another, and provides a useful theoretical model within which to explore fundamental limits to exploration and mapping. Another fundamental representation is based upon building the representation within some Cartesian space. In such a representation measurements can be identified with specific locations and orientations. Many other representations are of course possible, but the majority of algorithms build upon either a topological or metric representation.

In robotic exploration and mapping, the problem that lies at the core of SLAM algorithms is answering the question 'have I been here before?' This is also known as the 'loop closing' problem as identifying that 'I have been here before' enables loops to be

constructed or closed in the partially built map. Here we review two fundamental approaches to the problem. One approach is to resort to an 'oracle' that can be used to help the robot solve the disambiguation problem. Different forms of oracles have been employed. With an appropriate oracle, the SLAM problem can be solved deterministically. Deploying an oracle, however, is expensive, and might not always be feasible. Another approach is to exploit additional information describing the underlying environment. One obvious choice is the use of metric information. 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. Even with metric information, existing probabilistic, metric-based approaches have difficulties with large and complex environments. Many open questions still exist in SLAM with metric representations including how to exploit (other) knowledge of environmental properties when solving the SLAM problem. For example, how can we exploit *a priori* information about the total number of locations in the environment, or, about the probabilistic distribution of different kinds of locations while solving SLAM?

Structure of the report

The remainder of this report is organized as follows. Chapter 2 defines formally the exploration problem and environment and map representations. Chapter 3 reviews oracle-based approaches to the SLAM problem. Chapter 4 reviews oracle-less approaches which rely on metric information. Chapter 5 concludes the work and presents a discussion of open problems and possible directions for future research.

2 The mapping problem and spatial representations

This chapter begins by formally defining the problem to be investigated, followed by discussions of the robotic exploration and mapping problem and environment representations for the mapping problem.

2.1 Problem definition

Robotic exploration and mapping addresses the problem of acquiring spatial models (a 'map') of physical environments autonomously using mobile robot(s). This problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robot systems. The topological mapping problem addressed in this report can be defined formally as follows: Given a specific environment that can be modeled as an (embedded) graph, formulate a series of plans for the explorer(s)/robot(s) based on the local sensing information of the explorer(s), so that after carrying out the actions specified by the plans, the robot(s) will have constructed a topological map that is isomorphic to the underlying world being explored. Note that in terms of the edge traversals required by the robot, the lower bound cost of a topological mapping problem is M, where M is the number of edges of the graph-like world. This is trivially true, as a robot cannot map a graph-like environment without traversing each of the edges in the graph. A similar definition can be developed for metric mapping, where the goal becomes that of formulating a series of plans so that a metric map is constructed that is isomorphic to the true metric representation of objects in the environment.

In the development of such a mapping algorithm, a number of critical problems must be addressed, including

- How can the robot disambiguate locations during exploration?
- How can the robot use oracle(s) to solve the disambiguation problem?
- How can the robot solve the problem without oracle(s)?
- How should knowledge of environment properties be exploited during exploration?
- What kind of *a priori* information of environmental properties is required in order to enable the removal of the oracle while still solving the problem?

2.2 Mapping, localization and the SLAM problem

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, the work focused on solving mapping and localization problems independently. More recent research efforts address these two problems simultaneously.

Localization without mapping Much work has been done to estimate and maintain a robot's position and orientation within a complete representation of the environment. In this situation, it is typically assumed that the map is known with 100% certainty and the environment is static, i.e., the robot is given a map of its environment. The goal of the robot is to determine its position relative to this map [99]. Generally, the inputs to localization algorithms include the knowledge of the environment (a map) and measurements (sensor readings) of previous steps. A localization algorithm takes these inputs and generates the best estimate of the robot pose (position) within the environment.

Mapping without localization Early work in robotic mapping typically assumed that the robot's pose in the environment was known with 100% certainty and focused on incorporating sensor measurements into different map representations of the environment. This problem is also known as mapping with known poses [99]. The inputs to the problem include the set of all measurements (sensor readings), and the path of the robot defined through the sequence of poses occupied by the robot. A mapping algorithm takes these inputs and generates the best estimate of the map representations. The obvious problem of such mapping algorithms is the lack of a method for accommodating pose uncertainty.

Joint estimation (SLAM) 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. In [99] the SLAM problem is defined as a *simultaneous localization and mapping* problem, in which a robot seeks to acquire a map of the environment while simultaneously seeking to localize itself relative to this map. 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.

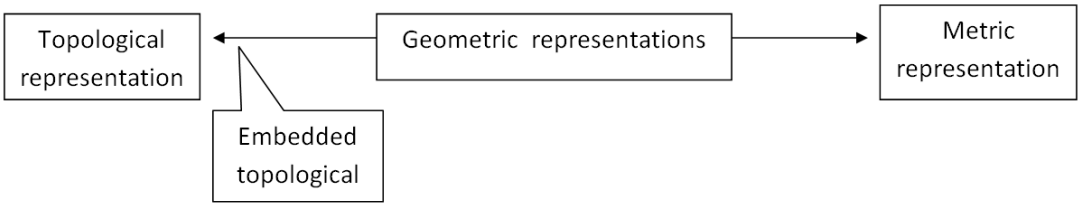


Figure 2.1: The range of spatial representations.

2.3 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., coordinates in a Cartesian representation) of the environment. In metric representations every component of the environment is embedded into the Cartesian space. While topological representations are concise, metric representations provide a detailed world representation but require more storage. These two paradigms can be considered as the two ends of a space of pose representations. At one end is the (pure) topological representation, and at the other end is the (embedded) metric representation. While topological and metric representations are the two traditional paradigms in the SLAM literature (as discussed in the later chapters), there exist other representation possibilities. These possible representations include the *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 the geometric representations some geometric information is maintained, but unlike in the (embedded) metric presentations the components in a geometric representation are not necessarily embedded into a Cartesian space. For example, in [3] the map consists of collection of segments, and the angles between pairs of segments are maintained. As another example, in [65] 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 pictorial illustration of the various representation possibilities including the embedded topological representation (discussed next) is given in Figure 2.1.

2.3.1 Topological representations

One fundamental 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. Environments in topological maps are typically represented as a set of significant places (vertices) that are connected via arcs (edges), i.e., an (embedded) undirected or direct graph (as will be made clear in later discussions). There exist, however, other graph-like representations of the environment. For example, in [70] 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. With a topological (graph-like) representation, map learning 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 [98]. 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.

Embedded topological representations

As will be made clearer in the next chapter, most of the topological representations used in the literature are embedded topological representations. There are various ways that the embedding can be defined. Here we introduce the definition of Dudek et al. [34, 36] as an illustrative and representative example of an embedded topological representation. This definition has been adopted by a number of subsequent research efforts reviewed in this report. The properties of this model represent typical properties of the general embedded topological representations. In [34, 36] the world is modeled as a graph embedding consisting of vertices, a set of edges between them and a local ordering defined on all edges incident upon each vertex. Specifically, the world is defined as an undirected graph $G = (V, E)$ with a set of vertices $V = \{v_1, ..., v_n\}$ and a set of edges $E = \{(v_i, v_j)\}$. The labels (if any) on vertices and edges of G are invisible to the robots, so that vertices and edges are not uniquely distinguishable to the robot. (In the literature this is referred to as an *unlabeled* or *anonymous* graph [22, 54].) G is embedded within some space in order to permit relative directions to be defined on the edges incident upon a vertex. More formally, the definition of an edge is extended to allow for the explicit specification of the order of edges incident upon each vertex of the graph embedding. This ordering is obtained by enumerating the edges in a systematic (e.g., clockwise) manner from some standard starting direction. An edge $e = (v_i, v_j)$ incident upon v_i and v_j is assigned labels p and q, one for each of v_i and v_j respectively. The lables p and q represent the ordering of the edge $e_{i,j}$ with respect to the consistent enumeration of edges at v_i and v_j respectively. p and q can be considered as general directions, e.g., from vertex v_i then pth exit takes edge e to vertex v_j. A route (path) can be specified as a sequence of edge labels such that the entry edge at a vertex is always the reference edge and the successive labels specify the exit edges (e.g., take the 3th edge on the right, then take the 2nd edge on the

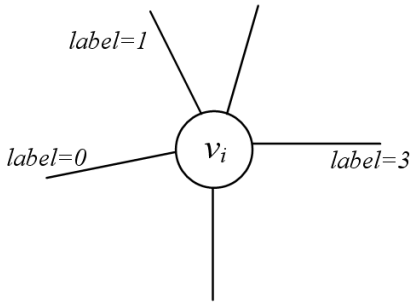


Figure 2.2: A vertex with fixed order (relative direction) of edges (exits) leaving the vertex.

right etc). Note that, as an unlabeled (anonymous) graph, the absolute edge ordering defined by the embedding is not accessible to the robot.

There are several properties of the model in [34, 36] that are worth noting. First is the embedding assumed. With the embedding, there exists an ordering defined on all edges incident upon each vertex. The ordering captures the relative direction (orientation) in which edges (viewed as exits) leave a vertex (Figure 2.2). As long as one edge is identified, the labelling or ordering of the others can be determined. Without embedding, for a vertex having d incident edges, there would be $d!$ ways of labeling the edges, in terms of their relative positions. This embedding assumption, together with the relevant assumption that a robot can enumerate edges in a consistent way, greatly simplifies many problems. To see the power of embedding, note that with the embedding a route (path) can be specified as a sequence of edge ordering (relative direction) with respect to the entry edge, e.g., 'take the 3rd exit on the right (with respect to the entry edge), upon arrival take the 2nd exit on the right'. Such a specification of route or movement would not be possible without an embedding (any other edge can be 'the 3rd right exit' with respect to the entry exit). As we will see in the next chapter, such a graph embedding is a key assumption in different topological representations. Another property of the model is that this graph-like representation is minimalist. In the model, edges are completely featureless and vertices are featureless except for the paths to other vertices. No spatial metric such as distance or orientation is required.

2.4 Summary

Robotic exploration and mapping addresses the problem of using autonomous robots to acquire spatial models of their physical environments. Solutions to this problem address 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*, the

problem of constructing an spatial representation (map) of the environment ('what does the world look like?'). In practice, if neither the pose nor the map is available, localization and mapping has to be solved concurrently, resulting in the *simultaneous localization and mapping* (SLAM) problem.

Various types of maps and environmental representations exist in the literature, from topological to metric. The choice of representation is often linked to the nature of SLAM algorithms, as will be made clear in later chapters.

During robotic exploration, one fundamental problem is 'have I been here' problem, i.e., closing cycles (loops). That is, to disambiguate the current place of the robot against known locations. One approach is to resort to an 'oracle' that can help robot solve the disambiguation problem. Another approach is to exploit additional metric information describing the underlying environment. Work in these two paradigms are reviewed in Chapters 3 and 4 respectively.

3 Exploring with an oracle

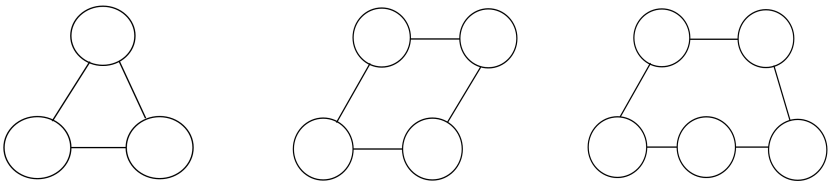
This chapter reviews notable work on robotic exploration and mapping with oracle(s). In such work a topological (graph-like) representation of the underlying world is typically assumed. A robot cannot distinguish locations of the graph-like world using its sensing alone. Therefore an 'oracle' that can help the robot solve the 'have I visited here before' problem is needed. Here 'oracle' refers to a tool or device that the robot or active agent can use to disambiguate locations during exploration. There are a number of ways to implement an oracle, resulting in oracles of different 'strengths' [1]. The relative strengths of different oracles are explored in this chapter. In the work reviewed in this chapter, oracles are simulated using immovable markers, movable markers, multiple markers and the like. ('Markers' are also termed 'pebbles', 'tokens' or 'beacons' in the literature.) Problems for some of the oracles are formulated for both undirected and directed graphs.

3.1 How necessary is an oracle?

An interesting and fundamental question about topological exploration with a mobile robot is: can a robot explore and map an arbitrary anonymous graph without any oracle? In their movable marker work [34], Dudek et al. discussed the problem and claimed that the answer is no. To see this, consider the graphs shown in Figure 3.1. Suppose a graph embedding such as that defined in [34] in which the vertices are all identical except the degree information (i.e., the number of incident edges), and at each location (vertex) the robot can only sense the degree information of the vertex. Whenever the robot enters a vertex, the robot cannot tell whether the vertex is a new vertex or is one of the vertices it had visited previously, since all the vertices (both the unknown and existing) have the same degree information. Moreover, the degree information of vertices connected to any vertex are all the same. All of the vertices thus appear identical to the robot, even if the degree information of arbitrarily large neighborhoods are taken into consideration by the robot. Thus if the robot were to explore the three different unknown environments shown in Figure 3.1 it would not be able to tell them apart, even though here we assume perfect robot motion and sensing in identifying a vertex and enumerating the incident edges.

Note that there are an infinite number of other graphs that are indistinguishable from

[1] For an early solution to partial mapping using a pebble-like oracle see the solution proposed by Hansel and Gretel [60] and see [1] for the string-based oracle that Ariadne communicated to Theseus.



(a) Single cycle graph of three vertices

(b) Single cycle graph of four vertices.

(c) Single cycle graph of five vertices.

Figure 3.1: Simple indistinguishable graphs. Each vertex appears identical to every other vertex. Note that there are infinite number of other graphs that are indistinguishable from these examples.

these examples – the robot would not tell apart any sized single cycle graph in which every vertex has degree two. In exploring both the graphs, the robot always observes a non-terminating sequence of '2-door rooms'. Moreover, the class of unexplorable environments without an oracle is much larger than the set of graphs shown here. Many graphs that contain symmetries will cause the zero-oracle robot to fail [34].

Although a deterministic solution is impossible to the topological SLAM problem, a probabilistic one is. This is illustrated in [33] which describes a probabilistic oracle-less approach for anonymous graphs. Without an oracle, for each current place the approach has to consider all possibilities, i.e., the current place may be a new place or one of the known places (i.e., a loop is closed), and as shown in the figure there is no way of collapsing this probability function without resorting to other external assumptions (or oracles). For a 3-vertex cycle, the algorithm generates infinite number of models each corresponding to a different sized single cycle. This oracle-less algorithm is reviewed later in the chapter.

The following sections of this chapter present discussions on different kinds of oracles. Starting from the single undirectional super-glue (immovable) marker, which is probably the weakest oracle, these include the single immovable marker, the single movable marker, multiple homogeneous markers and multiple distinct markers. Problems for some oracles are formulated for both undirected and directed graphs. These are discussed separately. Non-deterministic approaches employing impoverished oracles are also discussed later in the chapter.

3.2 Exploring with a super-glue marker

An interesting oracle is a '*super-glue*' marker, which is a single marker that the robot can use to uniquely mark (label) one of the (visited) vertices of the unknown environment. Specifically, this is a immovable marker which can be dropped but will remain in the

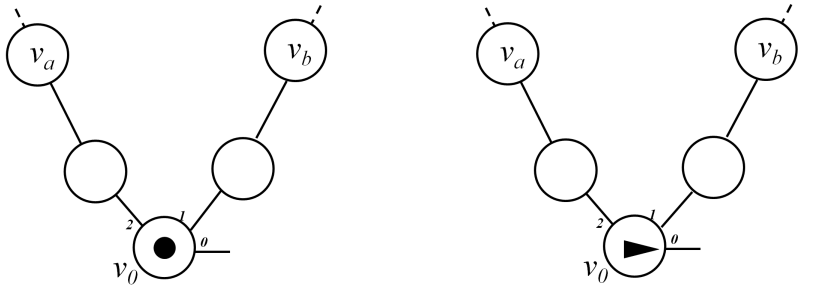
location during the exploration. Whether or not one super-glue marker oracle can solve the general exploration problem is an interesting problem, which, to the best of our knowledge, has not been formally addressed in the literature. Dudek et al. presented brief discussions of the problem in [34, 37]. In [34], the authors conjecture that if a single immovable marker is used, then the class of graphs that can be successfully explored is reduced, and the robot cannot explore all its environment successfully. Neither can it be sure that it has visited every vertex.

3.2.1 An undirectional super-glue marker

Perhaps the weakest oracle is an *undirectional* 'super-glue' marker, which can uniquely mark one of the visited vertices of the unknown environment. (Directional super-glue markers that can mark a specific edge leaving a vertex are also possible and these are considered below.) The challenge of using such a 'weak' oracle is how to disambiguate other places. First note that one super-glue pebble is potentially helpful. Assume a graph embedding as described in [34] (also described in Chapter 2), and suppose that the marker is dropped at some locations v_0. Now consider a path from a particular location within the underlying world to v_0. Given the ordering of edges at each location due to the graph embedding, the path can be represented as the sequence of edge orderings (labels) at each vertex along the path, and it is trivially true that there always exist *distinct* paths (in terms of door sequence) from different edges of the underlying world to vertex v_0 where the marker was dropped. That is, the (relative) door sequence(s) for different edges are different. (This is trivially true. Suppose on the contrary, two different edges of the graph have the same relative door sequence towards a particular door (edge) of v_0. Then this implies that starting from the particular door (edge) of v_0, by following the reverse relative door sequence both the two distinct edges can be reached. This contradicts to the fact that the reverse door sequence can determines only a particular door in the underlying world.) These distinct paths can be used to disambiguate different locations. Despite the existence of such distinct paths, however, an undirectional super-glue pebble does *not* provide sufficient information for the robot to distinguish the distinct paths. Consider the example illustrated in Figure 3.3(a). The robot cannot distinguish between the (distinct) path $v_a, ..., v_0$ and $v_b, ..., v_0$. The two paths have the same sequence of (relative) edge orderings, and only differ at the ordering of the entry edge at v_0. Given the assumption in [34] that the graph is completely anonymous including the edge ordering defined by the embedding, when entering v_0, it is impossible for the robot to determine the door (edge) of v_0 through which it entered.

3.2.2 A directional super-glue marker

Although an undirectional super-glue marker is insufficient, the mapping problem becomes solvable if the robot is able to determine the ordering (labels) of incident edges at v_0 and observe the ordering of the door it enters through when it enters v_0. In an



(a) An undirectional super-glue marker (b) A directional super-glue marker

Figure 3.2: Two kinds of super-glue marker.

anonymous graph, this information is available if the super-glue pebble is *directional* and the robot can sense this direction information.

Assume the robot drops the marker at v_0 and points the marker head toward one of the doors (edges), as shown in Figure 3.3(b). Then whenever the robot returns to v_0, by enumerating the doors and identifying the marked (pointed) one, the robot is able to distinguish different edges at v_0. Moreover, by remembering the label of the marked edge, the robot can infer the absolute ordering (labels) of all the incident edges at v_0 (including the edge by which it entered v_0). That is, a directional super-glue marker not only identifies a unique vertex in which it is dropped but it also provides the unique edge ordering at that vertex.

The directional super-glue marker algorithm

Assume the (undirected) embedded graph representation modeled by Dudek et al. [34] (also described in Chapter 2). Assume that the robot can drop the super-glue marker, positioning the marker toward one of the doors (edges). Also assume that the robot can move from one vertex to another by traversing an edge. 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 enables a robot to sense whether the super-glue marker is present at the current vertex. If the marker is present, the robot can also sense the direction of the marker, i.e., determine which door (edge) the marker points to. Here, direction is a pointer to a specific edge at the vertex. This direction specifies not only the first edge in a global enumeration to this vertex, but also sufficient information for the robot to assign a unique global enumeration of all edges leaving from this vertex. Note that in a planar environment this enumeration might be as simple as to enumerate the doors (edges) in a clockwise manner, but more sophisticated enumeration schemes are required for higher dimensional spaces

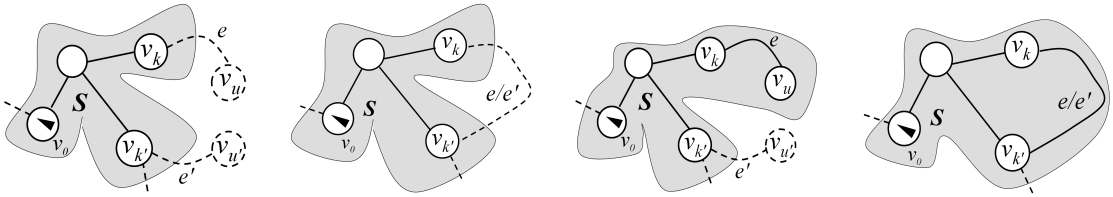
or spaces lacking a gravity-like reference frame. With *edge-related* perception, a robot can determine the relative positions (ordering) of edges incident on the current vertex. Specifically, the robot can identify the edge through which it entered the vertex and, by following the pre-defined ordering convention, assign a relative label (index) to each edge in the vertex representing its current local edge ordering. Note that this local edge ordering is not, in general, equal to the unknown ordering specified by the embedding (which is not accessible to the robot), but rather is a rotation of it. When the directional super-glue marker is dropped at v_0, the robot remembers the label of the door (edge) that the marker points to. During exploration, the robot also remembers all of its actions. Specifically, if the robot has performed steps $0, 1, ..i$, the memory of the robot contains the sequence of information at each step. For the *i-th* step, it remembers marker sensing at the step, the order of edges incident on the vertex visited at step i. By "memorizing" the motion sequence, the robot is able to retrace any previously performed motion.

Similar to the movable marker algorithm of [34] (reviewed below), the directional super-glue marker algorithm proceeds by incrementally building a known map out of an explored subgraph S of the underlying graph G. As new vertices are encountered, they are added to the explored subgraph S, and their outgoing edges are added to U, the set of edges that lead to unknown places and thus must be explored. Initially $S = \{v_0\}$ where v_0 corresponds to the initial location of the robot and where the robot will drop its pebble. Incident edges at v_0 are the initial elements of U. The robot enumerates incident edges at v_0 and sets labels (on S) based on the enumerated edge ordering. The robot then drops the directional marker at v_0, pointing the marker toward one of the doors (edges) and remembers the label of the door.

One step of the algorithm consists of selecting (and removing) an unexplored edge $e = (v_k, v_u)$ from U, traversing to the known vertex v_k and then following e to the unknown end vertex v_u, as shown in Figure 3.3(a). Upon arrival at the unknown end v_u, the robot needs to determine 'have I been here before'? Specifically, the robot must answer:

1. Whether or not v_u corresponds to a known location in S?
2. If v_u corresponds to a known location $v_{k'}$, then which incident edge of $v_{k'}$ does edge e correspond to?

In the following discussions, determining (1) and (2) are referred to as 'place validation' and 'back-link validation' respectively. Algorithms with different oracles conduct the validations in different ways. Here the validations are conducted by disambiguating edge e against (other) unexplored edges in S. Each (other) unexplored edge $e' = (v_{k'}, v_{u'})$ which is incident on a known vertex $v_{k'}$ is considered as a potential loop-closing hypothesis. That is, it is hypothesized that $e = (v_k, v_u)$ and $e' = (v_{k'}, v_{u'})$ correspond to the same edge (thus the robot has entered $v_{k'}$ from v_k via e'), as shown in Figure 3.3(b). (Note that certain hypotheses can be trivially rejected, but we ignore those here for simplicity of exposition.) For each hypothesis the (shortest) path $v_{k'}, ..., v_0$ on S is computed. The



(a) Traverse to v_u via e

(b) A hypothesis

(c) All hypotheses fail.

(d) Hypothesis is confirmed

Figure 3.3: Directional super-glue 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.

path is represented as a sequence of (relative) edge orderings at each vertex visited along the path, including the ordering at $v_{k'}$ (relative to the known ordering of e' at $v_{k'}$), and the ordering at v_0 (relative to the known ordering of the pebble-marked edge). The robot then attempts to traverse this path, which would start from $v_{k'}$ and lead it to v_0 via the expected entering edge if the hypothesis holds, i.e., if the robot has entered $v_{k'}$ from v_k via e'. The algorithm distinguishes three possibilities:

1. The pebble is encountered at some point along the execution of the path prior to completion.

2. Upon completion of path execution, the pebble is not present, or it is present but the entering edge does not match the expected entering edge in v_0.

3. Upon completion of path execution, the pebble is present and the entering edge matches the expected entering edge in v_0.

In case (1) the hypothesis is rejected. As $v_{k'}, ..., v_0$ was the (optimal) shortest path to v_0, the robot should not have encountered the pebble 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 not arrive from the correct entry edge. Once a hypothesis is rejected, the robot retraces its steps by the reverse edge sequence (to v_u), and then the next hypothesis of e is tried (if any). Finally, in case (3) the hypothesis is confirmed and no other hypotheses of e is tried.

If all the hypotheses of e are rejected, then the unknown location v_u does not correspond to a known vertex in S and can be added to S as a new vertex. Edge e is also added to S as an explored edge, augmenting S by one edge and one vertex (Figure 3.3(c)). Other edges incident on v_u are added to U. Note that now the algorithm is free to set the ordering (labels) of e and the other incident edges at this new vertex, representing the enumerated edge ordering of the edges at the new vertex (e.g., e is the 0'th edge and the

others are ordered accordingly). If the hypothesis 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 S is augmented by the edge $e/e' = (v_k, v_{k'})$ (Figure 3.3(d)). The algorithm terminates when the unexplored edge set U is empty. The algorithm is outlined in Algorithm 1.

Algorithm 1: Mapping with a directional super-glue marker

```
Input: the starting location v_0 in G; a directional super-glue marker
Output: a map representation S that is isomorphic to world G
the robot drops the directional super-glue marker at v_0;
S ← v_0; // initial S;
U ← 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;
    for each edge (hypothesis) e' = (v'_k, v'_u) in U do
        compute the shortest path {v'_k, ..., v_0};
        the robot traverses path {v'_k, ..., v_0};
        based on the sensory information obtained during the traversal do
            case (1) or (2)
                the robot retraces to v_k;
                reject the hypothesis and continue;
            case (3)
                confirm the hypothesis and exit the loop;
    // now do augmentations;
    if a hypothesis is confirmed then
        add edge e/e' = (v_k, v_{k'}) to S;
    else // all the hypotheses are rejected;
        add e and v_u to S;
        add other edges in v_u to U;
return S;
```

It is provably correct that when U is empty, S is isomorphic to the underlying graph-like world G. The key to the correctness of the algorithm is the justification that no two different edges of the underlying world have the same path from the edges to vertex v_0 where the super-glue marker was dropped (as shown earlier). Thus, upon completion of the path $v_{k'}, ..., v_0$ for hypothesis of e, if the robot senses the marker and the entering edge matches the expected edge, the hypothesis for e is confirmed. In the algorithm the explored subgraph S is maintained and augmented in a way that is similar to that in the movable pebble algorithm by Dudek et al. [34], and thus the correctness follows the

correctness proof given in [34].

Consider the complexity of the directional super-glue marker algorithm. Certain steps of the algorithm are executed mechanically (e.g., edge traversals) while others are executed electronically as the robot reasons (computes) about the model. As the time constant associated with physically moving a robot is considerably larger than that associated with a computational step, mechanical complexity is the limiting factor in the performance of the algorithm [34]. For the work here and other oracle-based graph exploration algorithms in this chapter, the mechanical time complexity of the task is the critical measurement of performance of a mapping algorithm. Assuming one mechanical step for the traversal of one edge, the main cost comes from the need for the robot to traverse the path $v_{k'}, ..., v_0$ and then traverse back to the current location in order to solve the key problem of 'have I visited here before'? This traversal is required for each edge hypothesis before an edge is added into S. Each (other) unexplored edge in S is a hypothesis, so the number of hypothesis is bounded by the number of edges in the graph. The traversal length of each $v_{k'}, ..., v_0$ is bounded by the number of vertices in the graph. Thus the algorithmic cost of mapping a graph $G(V, E)$ in terms of edges traversed by the robot (mechanical complexity) is bounded by $O(M^2N) \leq O(N^5)$, where M is the number of edges of G and N is the number of vertices of G.

Enhanced directional super-glue marker algorithm In practice the exploration cost can be reduced in a number of ways. For example, if the robot can detect that a hypothesis is invalid before completing the path of the hypothesis, then the robot does not have to complete the path, resulting in considerable savings. (Upon completion of the wrong hypothesis the robot either cannot find the marker, or otherwise did not arrive from the correct entry edge.) Given the enumeration ability of the robot, cost reduction can be potentially achieved by associating each door sequence with the expected 'degree sequence'. That is, for each hypothesis we compute both the relative door sequence of $v_{k'}, ..., v_0$ as described above, and the expected sequence of degree (number of edges) of each vertex that the robot should observe during traversal of the hypothesized path $v_{k'}, ..., v_0$. During execution of the path $v_{k'}, ..., v_0$, the robot senses the actual degree of each location it enters and compares with the expected degree associated with each vertex of the path. If the sensed degree at a location does not match the expected degree information, then the robot knows that it is deviating from the path $v_{k'}, ..., v_0$ and therefore the hypothesis is invalid. The robot can now terminate the traversal of the path immediately and try the next hypothesis (if any). Obviously the performance improvement depends on the homogeneity of the underlying graph. This approach would be of no help in the extreme case when the graph is completely homogeneous.

3.2.3 Summary

A single undirectional super-glue marker is not sufficient for mapping an embedded undirected graph deterministically, while a directional super-glue marker is sufficient. This

Oracle		Underlying graph-like world	
		undirected graph	directed graph
A super-glue marker	undirectional	no	no
	directional	yes. $O(M^2N)$	not known, probably no

Table 3.1: Solvability and the known cost of topological exploration with a super-glue marker. Note that the trivial lower bound for the topological exploration and mapping problem is $M = |E|$.

section described a directional super-glue marker algorithm and the enhanced version of it, which works for undirected graph in which the robot can retrace its steps. Some experimental results for the directional super-glue algorithms are shown later in the chapter, where the results are compared with that of the other oracles discussed in the following sections.

It is trivially true that a single undirectional super-glue marker is not sufficient for mapping an (embedded) directed graph deterministically. (Probably the simplest way to prove this is to observe that every (unsolvable) undirected graph can be transferred to a directed graph by replacing each edge in the undirected graph with two directed edges of opposite directions). Moreover, we conjecture that a single directional super-glue marker is not sufficient for mapping a directed graph deterministically. In a directed graph, the robot cannot retrace its steps by executing reverse path, which is the building block of the directional super-glue algorithm described above. Employing a directional super-glue marker on directed graphs is an interesting open problem. Solvability of the mapping problem using a super-glue marker under different conditions are summarized in Table 3.1.

An interesting question is how a single *movable* marker affects the solvability and cost of the problem. This is discussed in the next section.

3.3 Exploring with a movable marker

Using movable marker(s) as an oracle to help deal with uncertainty has been explored in earlier work including [86, 14, 13, 92]. Rabin first proposed the idea of dropping pebbles to mark nodes [86]. This suggestion led to a body of work that considered the searching capabilities of a finite automaton supplied with pebble(s). Blum and Sakoda [14] consider the question of whether a finite set of finite automata can search a 2 or 3-dimensional obstructed grid, which is also termed a 'maze' in the paper. A maze is defined as a finite, connected, obstructed checkerboard graph. It can be viewed as a subgraph of an embedded graph with the main difference that an automaton in a maze can distinguish North, East, South, and West directions (i.e., has a compass). The paper showed that a single automaton with just four pebbles can completely search any 2-dimensional finite maze, and that a single automaton with seven pebbles can completely search any 2-dimensional infinite maze. They also prove that no collection of finite automata can

search every 3-dimensional maze. Blum and Kozen [13] improve this result by showing that a single automaton with two pebbles can search a finite, 2-dimensional maze. Their results imply that mazes are strictly easier to search than planar graphs, since they also show that no single automaton with pebbles can search all planar graphs. Savitch [92] introduces the notion of a maze-recognizing automaton (MRA), which is a DFA with a finite number of distinguishable pebbles. The work shows that maze-recognizing automata and $\log n$ space-bounded Turing machines are equivalent for the problem of recognizing threadable mazes (i.e., mazes in which there is a path between a given pair of nodes).

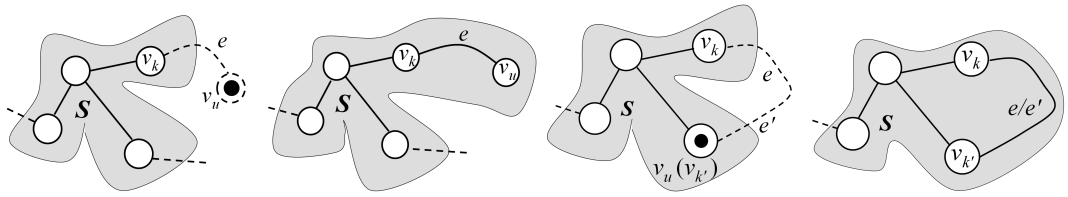
Depending on the topological representation adopted, the more recent work reviewed below can be broadly divided into those that model the environment as an undirected graph, and those that model the environment as a directed graph. We start with marker-based approaches on undriected graphs. These approaches are further divided into those that employ an undirectional movable marker and those employing a directional movable marker.

3.3.1 An undirectional movable marker

Work in this category represents the world as an undirected connected graph. One notable work is Dudek et al.'s single movable marker algorithm [34, 36]. There also exist a number of related papers that assume the same world and robot model or the variations of the models. For example, some work assumes an undirected *planar* graph. We start with Dudek et al.'s single movable marker algorithm.

The embedded topological representation assumed in [34, 36] is described in Chapter 2. It is assumed that a robot can move from one vertex to another by traversing an edge. The robot is equipped with an undirectional movable marker. 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 enables a robot to sense whether the marker is present at the current vertex. The *edge-related* perception is the same as that assumed in the super-glue marker case, i.e., the robot can determine the relative positions of edges incident on the current vertex in a consistent manner.

The algorithm proceeds by incrementally building a known map out of a known subgraph S of the full graph. As new vertices are encountered, they are added to the explored subgraph S, and their outgoing edges are added to U which is the set of edges that lead to unknown places and thus must be explored. A step of the algorithm consists of selecting (and removing) an unexplored edge $e = (v_k, v_u)$ from U, and validating the unknown end vertex v_u and the entry edge e. With a movable pebble, 'place validation' is carried out first, by the robot placing the marker at v_u and then visiting all potentially confusing vertices of the known subgraph S along edges of S, looking for the marker. (A known vertex in S is potentially confusing if it has unexplored edge(s) and has the same degree



(a) Marker 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 3.4: 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 marker is represented by $\bullet$.

as v_u.) There are two possible situations: the marker is found at one of the vertices in S, and the marker is not found in S. 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 subgraph S (Figure 3.4(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 the explored subgraph S by one edge and one vertex (Figure 3.4(b)). Other edges incident on v_u are added to the unexplored edge set U. If the marker is found at some vertex $v_{k'}$ of the explored subgraph S, then vertex v_u (where the marker was dropped) corresponds to the known vertex $v_{k'}$ where the marker was found (Figure 3.4(c))). In this case, 'back-link validation' is required, i.e., to infer the incident edge e' at $v_{k'}$ that edge e correspond to. The robot drops the marker at v_k and goes back to $v_{k'}$ along the shortest 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 the subgraph S and removed from U. In this case S is augmented by an edge (Figure 3.4(d)). The algorithm terminates when the set of unexplored edges U is empty. 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 $M = |E|$ is the number of edges in G and $N = |V|$ is the number of vertices in G. This cost comes from the need for the robot to go back to its known sub-graph and visit all of the locations there to solve the 'have I visited here before?' problem. The algorithm is outlined in Algorithm 2.

[34] has been extended by the authors and others. Closely related work include [35, 37, 27, 25, 91, 33, 38, 105, 106, 107]. Later work by Dudek et al. [37] assumes the same world model as in [34] and investigates two problems that are related to the exploration problem in [34]. The first problem is the 'Map Validation' problem. The

Algorithm 2: Mapping with an undirectional movable marker

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

```
1  S ← v_0; U ← incident edges in v_0; // initial S and U;
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      the robot drops the marker at v_u;
6      the robot traverses S searching for the marker;
7      if the marker is not found then
8          add e and v_u to S;
9          add other edges in v_u to S;
10     else // the marker is found in v_k';
11         do 'back-link' validation;
12         add edge e/e' = (v_k, v_k') to S;
13     the robot (goes to v_u and) picks up the marker;
14 return S;
```

second is the 'Self-location' problem. In the Map Validation problem, the robot is given a map of the graph-like environment. The map is a graph of the same form as the one computed by using the exploration algorithm in [34]. The robot is told which map vertex is its current location, and the correspondence between one map edge incident on the current map vertex and a physical exit form the current physical vertex, i.e., 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 (which is akin to an exploration task, i.e., using marker for location disambiguation). During validation, whenever the information the robot senses about the real world does not match the expected information modeled by the map, the validation fails. The sensed information are the degree of the node visited, and the presence or absence of a marker at the current node. The validation algorithm requires $O(N^2)$ moves 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, discarding hypotheses

which are found to lead to inconsistencies during exploration. The Self-location algorithm is much like the exploration algorithm described in [34] in terms of the physical steps of the robot, except that additional data structures are maintained as the robot moves. For each hypothesis, consisting of an initial pose depicted on the map, a correspondence is maintained between world vertices/edges and map vertices/edges as the robot carries out the exploration algorithm. Whenever the information the robot senses about the real world (marker and degree information) does not match the information modeled 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. [27] also follows the world model introduced by [34] and conducted a competitive analysis for the performance measurement of different strategies, including the exploration algorithm in [34]. The concept of competitive analysis was first introduced to deal with unknown future events of online problems [93]. The main idea is to evaluate how good a strategy operating under incomplete information is by comparing it with the optimal solution with complete information. In this approach, as in the authors' early work [28], 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 the initial position-orientation of the robot 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. With the terminology of competitive analysis, the single movable marker algorithm by Dudek et al. ([34]) is of competitive ratio $O(N)$. The paper shows that for the model in [34] and in the single marker case, the result of [34] 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 a 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 establish 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 marker is $\Omega(\log N)$.

Given the high potential cost associated with single robot exploration and mapping, Dudek et al. [38] extended the concept of a single robot exploring a graph-like world to the case of multiple robots. [38] sketched how multiple mobile agents might exploit the

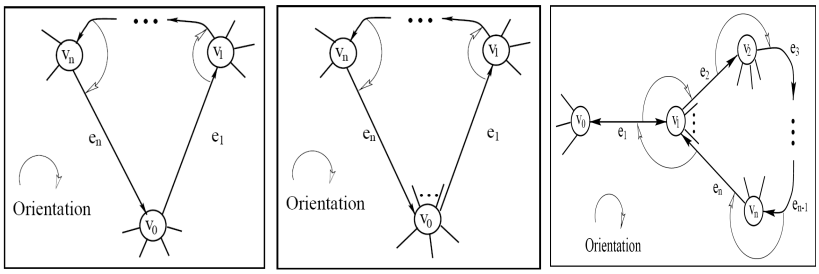
algorithms developed in [34, 36] to explore in a coordinated fashion. A critical restriction of [38] was that the individual members of the robot team were limited to communication when they were in the same graph node, and thus multiple robot exploration requires coordinated exploration and representation map merging in order to be effective. Wang et al. [105] formally developed the approach suggested in [38] and evaluated the performance of the two robot algorithm relative to that of a single robot exploring the same environment. This extension assumes the same environmental representation as described in [34, 36] and populated the world with two or more robots, each of which is equipped with its own undirectional movable marker. The robots can sense and communicate with each other only when they are at the same physical location (the same vertex). Also in the model all of the robots operate in parallel. The parallelism does not assume a global clock, or that distance or velocity information is available. Thus the only 'clock' robots have access to is the number of edges that they themselves have traversed.

In [38, 105] the joint exploration is achieved through alternating phases of independent exploration by the individual robots and coordinated merging of the independently acquired partial world representations. At any time the robots retain a common representation of some part of the world (the commonly known subgraph) S_m that evolves over time as well as independent information regarding other parts of the world. As successive iterations of the independent exploration and merging phase take place, S_m grows monotonically until it is isomorphic to the entire world map. The algorithm proceeds by having all of the robots start at a single location with a common local edge ordering (the initial definition of S_m), and then partitioning the unknown edges leaving the known world S_m between the robots. With their assigned edges each robot explores independently using the exploration algorithm described in [34]. After exploring for a previously agreed-upon interval defined in terms of the number of edge-traversals, the robots return to a commonly known and agreed-upon location to merge their individually acquired partial world representations. The merged map (augmented S_m) is then shared between the robots becoming the new commonly known representation S_m and the remaining unknown edges of S_m are re-partitioned between the robots for the next phase of independent exploration. The entire algorithm repeats until the environment is fully explored, that is, when there are no unexplored edges in the (merged) known map. The merging process takes two partial maps and involves disambiguating possible confusions between the two partial maps. One of the partial maps is chosen as the *base map* which is augmented with information in the other map. Similar to the exploration strategy the disambiguation of possible locations involves choosing an unmerged location in the other partial map, dropping a marker at this location and searching the base map for the marker. If the marker is found in the base map, then this location corresponds to a known place in the base map. If the marker is not found, the location is new and should be added to the base map. In either case, additional information has been added and more of the base map has been augmented. Empirical evaluation conducted in [105] show 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 [38, 105] were also investigated by Wang et al. [106, 107]. [106] investigates two enhancements. One enhancement considers the order in which potential places are explored and another considers the exploitation of local neighbor information to help disambiguate possible locations. Observing that in the original algorithm ([34, 105]) the 'closest-first' selection of the new place to explore generates repeated traversals to the same vertices for some environments, the paper examines an alternative 'breadth-first' exploration, i.e., explore all unknown edges of vertex v first, then explore all unexplored edges of neighbors of vertex v, and so on. The goal is to try to maintain a compact, fully explored region of the graph. To obtain more information to describe a vertex (and therefore more likely it is that disambiguation can be conducted without mechanical cost), the paper also extends the *signature* definition of a vertex. As defined by Kuipers et al. [69], a (perceptual) signature of a place is the set of landmarks which are 'visible' (i.e., which may be perceived) at a place, along with any other (local) identifying characteristics. In the original algorithm the degree of a vertex is considered as a instance of a signature. Here the signature of a vertex is extended to also include the neighborhoods of a certain distance, similar to the extended signature definition proposed in [32, 33] (reviewed later). Both the enhancements can provide a cost reduction over the original algorithm for some environments. Another work by the authors [107] examined a particular issue related to multiple robot exploration - the problem of how to prioritize certain tasks within the exploration process. The work extended the multiple robot exploration strategy described in [38, 105] by allowing the robots to explore in a 'lazy' fashion in which harder tasks are delayed until easier tasks are completed, taking advantage of the fact that exploration often becomes easier as more of the world is explored. It is shown that exploring in a lazy fashion can provide a cost reduction in exploration effort over the original algorithm for some environments.

There also exists some work that assumes a variation of the (undirected) graph-like world. Rekleitis and Dudek presented in [91] an algorithm for exploring an undirected *planar* graph. This constrained assumption permits mapping and exploration to take place much more efficiently than the algorithm in [34]. By exploiting the fact that a connected planar graph can be decomposed into a set of cycles which are called *'ears'*, the paper proposes an exploration algorithm with typically linear cost. In contrast to the technique in [34] where a single vertex is added at a time, here a closed path (ear) is added each time. The single undirectional movable marker is used in order to mark the starting node of the explored ear. At every 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. Upon arrival at a new vertex, the robot has an orientation set (e.g. clockwise) which determines the next edge to be traversed. For every new vertex visited, a node is instantiated and added to the new ear. When the robot arrives at the marked vertex it knows the number of edges traversed p, but it has no method of knowing the 'relationship' of the incoming edge



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

Figure 3.5: Three cases in exploring ears. Courtesy of Dudek et al. [91].

and the outgoing edge by which it started the exploration. To determine this relationship, which is required for connecting the newly explored ear to the explored subgraph, the robot methodically traverses the ear in the opposite direction. First the robot picks up the marker and backtracks to the previous vertex visited where it drops the marker, then the robot continues backtracking until it either reaches the marked vertex or it has performed additional p edge traversals. There are three different cases depending on the topology of the explored graph: 1) *Empty Ear*: After additional p traversals the robot found the marker. In that case the robot has explored an empty ear and the incoming edge is adjacent to the outgoing edge (see Figure 3.5(a)). There are no nodes inside such an ear as at every node the immediate neighbor was selected. 2) *Non Empty Ear*: After additional p traversals the robot has not found the marker. In that case the robot has explored an non-empty ear and there is a number of edges between the outgoing edge and the incoming edge (see Figure 3.5(b)). A sequential one-step search of every edge adjacent to the vertex is then conducted in order to reveal the number of edges between the outgoing edge and the incoming edge. 3) *Isthmus*: The marker is found after additional $p-2$ steps. In that case the incoming edge is identical to the outgoing edge (see Figure 3.5(c)). The explored path encloses a subgraph that has a single connection to the rest of the graph via the starting vertex. After a new ear is constructed and the incoming and outgoing edges are known the explored subgraph is updated using a recursive procedure which merges the newly explored vertices with the existing ones. When two vertices are merged their neighbors are merged as well. In [34] a (planar) graph representing the Toronto underground complex was fully explored with 5134 steps. The same graph is explored using the approach here and the resulting number of steps was reduced to 433. The authors justify that the efficiency of their approach derives from the fact that it is specific to planar graphs only. Note that in this work the marker, together the forward and backward edge sequences on the ear generate an 'invisible string' on the traversal path. Crossing the string at different points signifies loops of different kinds.

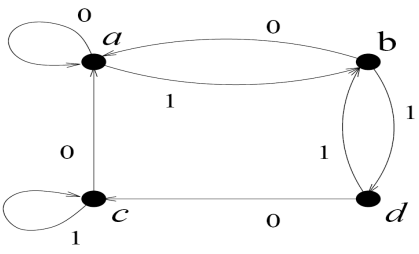
3.3.2 A directional movable marker

There are few results on exploring undirected graphs with a directional movable marker. As pointed out in [37], a directional movable marker is in general more powerful than an undirectional movable marker. [37] gives an example of improving the single undirectional movable marker algorithm in [34] using a directional movable marker. With a directional movable marker, the robot always drops the marker in the direction of the entry edge at each newly explored location v_u. Then whenever the robot finds the marker at a known vertex $v_{k'}$, by enumerating the edges and identifying the marked one, the robot can trivially infer the ordering (label) of edge e' of $v_{k'}$ that edge e corresponds to. Thus the 'back-link validation' process in the original algorithm (step 11 in Algorithm 2) is avoided. Without a directional marker, this process is necessary. With an directional movable pebble, the exploration cost is still $O(MN)$ but the constant is expected to be reduced.

Deng et al. [25] also adopts the same undirectional graph representation introduced in [34] and investigates the map validation problem using a directional marker. In contrast to the map validation problems described earlier, here the assumption about the marker is modified such that the robot is equipped with a single *directional* marker that can be put down at an edge of the graph world G (in the direction of traversal) and picked up later as needed. Another different assumption is that the given map 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* [62] of the augmented map M one by one, mimic the action on the environment graph, and compare the local observations observed (degree of the node visited and the presence or absence of a marker at the current node or edge) during tracing with the information on the map. The key observation is that although the sequence of these local observations for a single face by itself does not uniquely identify the topological structure of the face, the collection of all the observation sequences of the faces uniquely identify the topologies and verify the map. Initially, the robot is positioned at a vertex and oriented along one of the edges. The edge marker is dropped on the edge in the direction of the orientation of the robot. During tracing, when the edge the robot is about to traverse contains the edge marker in the direction of the traversal, the robot has completed tracing a face. The total number of edges traversed by the robot during the algorithm is at most 4 times the total number of edges of the map. The algorithm does not work when the map is not embedded in the plane.

3.3.3 Exploration on directed graphs

There also exists marker-based work that models the world as a strongly-connected directed graph (e.g., [11, 10]). Exploring a directed graph using a movable marker is more difficult than exploring in an undirected graph. In contrast to the case of an undirected graph, on a directed graph the robot cannot retrace its steps to retrieve the marker or explore. Specifically, once a robot has left a known portion of the graph, it might not know



(a)

homing sequence $h = \begin{cases} \text{Lewis:} & 0r1r \\ \text{Clark:} & r0r1 \end{cases}$

start node	output	end node
a	$TTST$	b
b	$STST$	b
c	$STST$	b
d	$STTT$	c

(b)

Figure 3.6: Illustration of a two-robot homing sequence and a lead-lag sequence. One robot Lewis executes $0r1r$, and the other robot Clark executes $r0r1$ (r means rest without move). h is a homing sequence because for each output sequence there is a unique end node. (Note that the converse is not true.)

how to return. This problem would vanish if there were a 'reset function', i.e., a *returning oracle* which when queried returns the robots to a particular starting node. In practice, unfortunately, robots exploring a directed graph do not have access to a 'returning oracle'. [11] and [10] thus seek to implement and learn a weak returning oracle, which can help the robot return back to some known vertex which can be 'recognized' (using some additional operations). [11] implements this idea using two robots and learns a two-robot *homing sequence*. This paper argues that exploring with two robots is more powerful than one robot with pebbles. (Later work [10] also learns the weak returning oracle but uses one robot with a pebble.) Realizing that a real returning oracle is not possible and cannot be learned, [11] suggests an alternative technique: a new type of *homing sequence* for two robots. Intuitively, a homing sequence is a sequence of (synchronized) actions by the robots on the graph, which is denoted by a sequence of edge labels traversed by the robots. The sequence of actions is a lead-lag sequence by performing which the robots follow the same path but at different speeds. The observed output is a sequence of T (together) and S (separate). The paper shows that every strongly-connected directed graph has a two-robot homing sequence. The main property of a homing sequence is that if the same output is observed, the robots will always be at the same final destination, i.e., the observed output uniquely determines the final node reached in G. An example of a two-robot homing sequence is shown in Figure 3.6. The paper first shows that given a homing sequence h, an algorithm can map G by maintaining several copies of the map and path, one of each corresponds to a different output sequence of h. The algorithm proceeds in stages. At each stage h is executed, and the algorithm explores a new edge in the map that corresponds to the output of h. The algorithm proceeds until eventually one of the maps is completed (i.e., has no unexplored edges). Exploring an edge in a map is conducted by robot A traversing an unexplored edge in the map, and the other robot

B tours the known portion of the map (looking for A). If the robots encounter each other then robot A's position is identified, otherwise A is at a new node. In practice, a homing sequence may not be available. Based on the justification that a two-robot homing sequence can be learned, the paper then presents a complicated algorithm in which the robots explore the graph and learn the homing sequence simultaneously. The algorithm maintains a candidate homing sequence h, and improves h as the robots explore. The paper develops a sequence of lead-lag actions by the two robots, which is used to verify whether the candidate home sequence h and the corresponding map are correct. If the map is shown to be in error, it is discarded and the sequence of actions is in turn used to improve h. The algorithm is shown to always output a map isomorphic to G and halt in $O(D^2N^5)$ steps, where D is the maximum out-degree. The paper also compares the computational power of two robots to that of one robot having a constant number of pebbles (markers). By introducing a class of graphs called *combination locks* within which the robot would lose its pebble (i.e., cannot find the pebble after dropping it) with high probability, the authors argue here that a single robot with a constant number of pebbles cannot efficiently explore strongly-connected directed graphs without *a priori* knowledge of the number of vertices N (whereas the two-robot algorithm in this paper requires no such *a priori* knowledge). The authors conjecture that the same holds when N is known.

The results of [11] motivates two questions: (1) How many undirectional movable pebbles (markers) are needed to learn graphs efficiently if N is known? (2) How many such pebbles are in fact needed if N is unknown? Later work [10] demonstrates that surprisingly few pebbles are needed in both cases. [10] shows that 1) If the robot knows N (or an upper bound $\widehat{N}$ on N), it can learn the graph with only one undirectional movable pebble 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 pebbles are both necessary and sufficient. The results disprove the conjecture in [11] that one robot with a constant number of pebbles cannot (efficiently) map a directed graphs even when upper bound on the number N is known. As in [11], the algorithm also emulates a 'weaker returning oracle'. The paper introduces the notation of *orienting procedures* which is analogous to the notion of two-robot homing sequences introduced in [11], which guides the robot around the graph and ultimately leaves the robot at a vertex it "recognizes". The paper first introduces an important technique: *path compression*. The robot executes this subroutine (using the pebble) to map subgraphs of G that are visited by closed paths known to the robot. Assume the robot is at node v and is given a closed path in G that starts and ends at v. The path visits a subgraph G_{path} of G. Since the path may visit the same vertices several times, G_{path} is not necessarily a simple cycle. In the path compression procedure the robot uses the pebble to identify repeated vertices on the path and construct a graph M isomorphic to G_{path}. For the path of length k, the procedure maintains a list of length $k+1$ where ultimately the ith entry in the list will identify the ith vertex occurring on the path in G. Initially, the list is (w_0, Λ, ..., Λ, w_0), where w_0 corresponds to v and Λ means "unidentified". The goal of the robot is to replace all unidentified entries with vertex names. The algorithm proceeds in stages, each starting and ending with the robot and

the pebble at the v. For example, in the 0'th stage, the robot drops the pebble at vertex v and follows the entire closed path; for each i such that the robot observes the pebble after i steps, the robot replaces the ith entry in the list with w_0. The paper then shows how to learn (augment) the map given the weak orienting procedure. Similar to the algorithm in [11], the robot maintains several copies of map, each corresponding to one output of the orienting procedure. In each stage one of the maps is learned (augmented) with an edge. One of the differences is that in learning the map corresponding to the output, the algorithm uses repeated execution of the orienting procedure to identify closed paths, and then uses path compressing techniques on the closed path to augment the map. The paper then introduces complicated strategies of learning the graph while building an orienting procedure. Similar to the algorithm in [11], a candidate procedure is maintained which is improved during exploration when a verification test of the map fails. The algorithm runs in time $O(\widehat{N}^2N^6D^2)$. The paper poses as open question whether this running time can be improved, either for the general case studied here or for special cases of interest. The above algorithm requires *a priori* knowledge of N. The paper also shows that there exists a (deterministic) algorithm that map any N node graph in polynomial time using at most $\Theta(\log\log N)$ undirectional movable pebbles without *a priori* knowledge of N.

3.3.4 Summary

An undirectional movable marker allows a robot to map an undirected graph-like world deterministically at cost $O(MN)$. This is in contrast to the case of a single immovable (super-glue) marker in which an undirectional marker is not sufficient while a directional immovable marker suffices with cost of $O(M^2N)$.

Similar to the case of a directional super-glue marker, a directional movable marker is more powerful than its undirectional counterpart, as the robot can benefit from the direction information provided. With such an oracle, the exploration cost is still $O(MN)$ but the constant is expected to be reduced (this is illustrated experimentally later).

Work on directed graphs show that exploring on a directed graph is more challenging. For a directed graph a single undirectional movable marker is not sufficient unless the number of nodes N (or an upper bound $\widehat{N}$ on N) is known *a priori*. In this case the robot can map the world using one movable pebble (marker) at the cost of $O(\widehat{N}^2N^6D^2)$. Whether a single directional movable marker can solve the mapping problem in directed graphs remains an open question.

Solvability and the known cost of exploration with a single movable marker under different conditions are summarized in Table 3.2, together with that for the super-glue marker discussed in the last section. The next two sections discuss exploration with multiple markers.

Oracle		Underlying graph-like world	
		undirected graph	directed graph
A super-glue marker	undirectional	no	no
	directional	yes. $O(M^2N)$	not known, probably no
A movable marker	undirectional	yes. $O(MN)$	no (unless N is known)
	directional	yes. $O(MN)$	not known

Table 3.2: Solvability and known cost of topological exploration with a marker. Note that the trivial lower bound for the topological exploration and mapping problem is $M = |E|$.

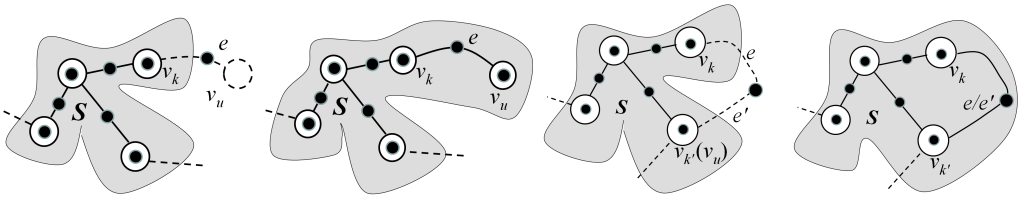
3.4 Exploring with multiple homogeneous markers

Given the exploration power of a single marker, it is interesting to consider the power of multiple homogeneous markers in exploration. (The power of multiple distinct markers is considered in the next section.)

3.4.1 Undirectional homogeneous markers

Perhaps the simplest multiple marker oracle occurs when each marker is an undirectional super-glue marker. If we assume that no degenerate edges exist in the world (multiple edges connecting the same two vertices) and that the markers can be dropped at vertices, then three such markers are sufficient to solve the SLAM problem deterministically (although as shown earlier, one is not). The robot drops one marker in its starting location (v_0), and then moves down one edge from v_0 and drops two markers there (call this v_1). The robot can now apply the directional super-glue marker algorithm where the vertex with one marker in it is v_0, and determining which edge is the marked edge can be established by traversing each of the edges in v_0 looking for v_1 in which two markers are dropped. If we assume the markers can be dropped at both vertices and edges, then two such pebbles are sufficient: one marker is dropped at v_0 and the other marker is dropped at one of the edges.

This requirement that no degenerate edges exist in the graph can be relaxed if more markers are available. Here we describe an algorithm for mapping a graph-like world with $N + M$ homogeneous super-glue markers that can be dropped at both vertices and edges. 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 'toeless footprints' on the underlying graph during exploration. For each current location, the footprints oracle answers 'have I been here before?', although it does not tell which visited location the current location is. Similar to the single marker algorithms described above, 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 marker/footprint at e). If v_u has not been visited before, i.e., v_u contains no marker



(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 3.7: Mapping with undirectional homogeneous markers (footprints). Markers/footprints are represented by •.

(Figure 3.7(a)), the robot drops a marker at v_u. Both e and v_u (foot-printed now) are then added into S without additional validations (Figure 3.7(b)). If v_u has been visited before, i.e., v_u contains a marker (Figure 3.7(c)), then both 'place validation' and 'back-link validation' are needed. The newly dropped marker/footprint on e is exploited in identifying the edge e' leaving v_u, which corresponds to e. Validations are 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 vertex that has an *unexplored* edge which has now been *foot-printed*. 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 3.7(d)). While the cost is $O(MN)$ as the robot may 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 needs for validations. The algorithm is outlined in Algorithm 3. A similar 'toeless footprints' algorithm is described by Deng and Mirzaian in [26].

Deng and Mirzaian also described in [26, 27] an algorithm for a robot to map an undirected *planar* graph in the toeless footprints model 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 is called a *rightmost DFS-tree* in the paper. The non-tree edges are called back-edges. The crucial property of a rightmost DFS-tree is that all the back-edges appear on the right 'shoulder' of the tree (Figure 3.8(a)). This forces a generalized nesting (or parentheses) structure among the back-edges. This property is exploited in the algorithm. If the robot moves from node v to node u and finds that there is no previous footprint on node u, then edge (v, u) becomes a tree-edge with $v =$ *parent(u)*. In this case u is given a new name and is added to the partial map. However, if there is a previous footprint on u, then (v, u) is a back-edge. In this case u could be

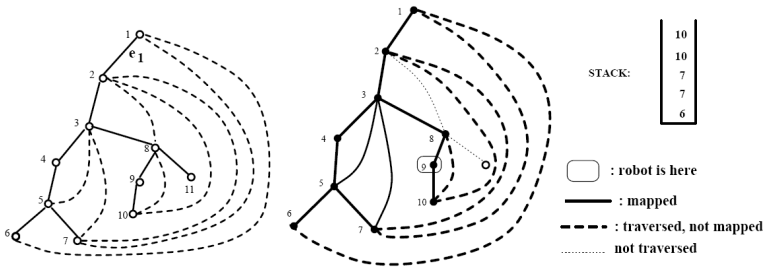
Algorithm 3: Mapping with homogeneous markers (footprints)

Input: the starting location v_0 in G; $M + N$ homogeneous markers
Output: a map representation S that is isomorphic to world G

```
1  the robot leaves a marker (footprint) at v_0;
2  S ← v_0; U ← incident 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 S to v_k and then follows e to v_u (leaves a marker at e);
6      if v_u does not contain a marker then
7          the robot leaves a marker at v_u;
8          add e and v_k to S;
9          add other edges in v_k to U;
10     else // v_u contains a marker already;
11         robot searchs S looking for the vertex that has one more foot-printed edge;
12         add edge e/e' = (v_k, v_k') to S;
13 return S;
```

either an ancestor or a descendant of v with respect to the rightmost DFS-tree. Node u is a descendant of v if there was a previous footprint on edge (v, u) and is an ancestor otherwise. In case (u, v) is a back-edge, node u must be one of the nodes in the partial map. To determine which known node u is, the algorithm exploits the nesting structure of the back-edges by using a stack S. The first time a back-edge is traversed, its (known) starting end-point is pushed on the stack, and the second time a back-edge is traversed (when it already has a footprint), the algorithm pops from the stack to determine which node of the map matches with the other (known) end of the edge. For each edge traversed but not yet identified, one of its endpoints is put on S, and the other endpoint will be determined later. A snapshot of the algorithm is shown in Figure 3.8(b). The paper shows that with footprints the algorithm maps an unknown embedded planar graph by traversing each edge at most twice.

Having a footprints oracle is equivalent to having $M + N$ homogeneous markers. How about less markers? [26, 27] also shows that with non-trivial modifications, the methods can be applied for exploring such embedded planar graphs with only N homogeneous markers to be placed on nodes but not edges, i.e. having footprints at nodes but not on edges. The modified algorithm uses other operations to compensate for the absence of footprints on edges, at the cost of increasing edge traversals for each edge from two to four. Without footprints on edges, in the modified algorithm whether a back-edge is being traversed for the first or second time is disambiguated by probing every edge incident to v to find out whether other end nodes, at that instant, have markers / footprints or not, and proceeding accordingly. This procedure causes the robot to traverse the edges incident to v twice. The rest of the DFS traverses these edges twice more for a total of



(a) (b)

Figure 3.8: A rightmost-DFS traversal of a planar embedded graph. Courtesy of Deng et al. [27].

four edge traversals.

If the robot has at least $N(N+1)/2$ super-glue markers, then it can drop at each vertex a different number of markers, marking each visited vertex uniquely. This is equivalent to having N distinct markers, and the problem can be solved in cost $O(MD_{max})$ where D_{max} is the maximum vertex degree, as shown in Section 3.5.

3.4.2 Directional homogeneous markers

If the homogeneous markers are directional, i.e., the footprints have 'toes', then both an edge ordering at each visited node and the direction of the first-time traversal at each visited edge are available. With $N+M$ directional markers, the above footprints algorithm can be improved. Define the *edgeINFO* of a visited (foot-printed) vertex as an ordered set that represents the presence (P) or absence (A) of footprints on the incident edges at the vertex, enumerated starting from the edge pointed by the directional footprint on the vertex. When traversing to a foot-printed end v_u ($v_{k'}$), the robot senses the *edgeINFO* there. Suppose v_u ($v_{k'}$) has four edges and the sensed *edgeINFO* of it is $\{P, A, P, A\}$. Then only vertices in S whose *edgeINFO* (retrieved from S) are also $\{P, A, P, A\}$ might be $v_{k'}$ and thus need to be visited. Any vertex having different *edgeINFO* (e.g. $\{A, P, A, P\}$) could not be $v_{k'}$. If the robot also considers the direction of the footprints on edges, i.e., incoming (P_i) or outcoming (P_o), then more vertices in S might be disambiguated against $v_{k'}$: Vertices having *edgeINFO* $\{P_i, A, P_o, A\}$), $\{P_i, A, P_i, A\}$) or $\{P_o, A, P_o, A\}$) are further disambiguated if the sensed *edgeINFO* of $v_{k'}$ is $\{P_o, A, P_i, A\}$. The worst case cost is still $O(MN)$.

If the robot has at least $N(N+1)/2$ directional markers, which is equivalent to having N distinct directional markers, then linear exploration cost $O(M)$ can be achieved, as is made clear in the next section.

Oracle		Underlying graph-like world	
		undirected graph	directed graph
A super-glue marker	undirectional	no	no
	directional	yes. $O(M^2N)$	probably no
A movable marker	undirectional	yes. $O(MN)$	no
	directional	yes. $O(MN)$	not known
$M+N$ homogeneous markers (foot-prints)	undirectional	yes. $O(MN)$	not known
	directional	yes. $O(MN)$	not known

Table 3.3: Solvability and known cost of topological exploration with foot-prints oracle. Note that the trivial lower bound for the topological exploration and mapping problem is $M = |E|$.

3.4.3 Summary

This section reviewed the foot-prints oracle, which is composed of $M + N$ homogeneous super-glue markers. With such an oracle, the robot can drop one pebble at each vertex and edge the first time the vertex or edge is visited, and does not pick it up again. For each current location, the footprints oracle answers 'have I been here before?', although it does not tell which visited location the current location is.

Using $M + N$ undirectional or directional homogeneous markers (i.e., a footprints oracle without or with toes) on embedded graphs (both undirected and directed) has not been fully addressed in the literature. We described an undirectional homogeneous markers algorithm that can map an undirected graph with $O(MN)$ cost. This algorithm can be improved if the markers are directional.

Another interesting oracle that has not been fully investigated is multiple *movable* homogeneous pebbles (markers). One of the few results is proposed in [10], which shows that there exists a deterministic algorithm that map a directed graph in polynomial time using at most $\lceil \log \log N \rceil$ movable homogeneous pebbles.

The solvability and the known cost of exploring with a footprints oracle are summarized in Table 3.3, together with that for the immovable and movable single marker oracles discussed in the previous sections.

3.5 Exploring with multiple distinct markers

An oracle that consists of at least N distinct super-glue markers can not only solve 'have I been here before?' but can also answer 'exactly which vertex the current vertex refers to in the set of previously visited vertices?' and/or to define a global ordering on this vertex. In the literature exploring with such an oracle is also referred to as 'exploring labeled graphs' [22].

3.5.1 Undirectional distinct markers

Having N distinct super-glue markers that can be dropped at vertices is equivalent to having $N(N+1)/2$ homogeneous markers. Such an oracle answers 'exactly which vertex the currently visited vertex refers to', With such an oracle, the single movable marker algorithm can be greatly simplified and show considerably improved performance, as all the visited vertices are already marked with their 'own' marker and thus 'place validation' is never needed. (We call such an oracle a 'vertex paint can'). Since the markers are undirectional, when exploring to a painted (i.e., visited) place $v_{k'}$ the robot still must perform 'back-link validation' to infer the particualr edge of the current place that the entering edge correspond to. This can be done by traversing each of the (other) incident edges of $v_{k'}$ trying to find a painted (visited) neighbour vertex of $v_{k'}$ (there must be at least one such a neighbor vertex, otherwise $v_{k'}$ could not be painted before). Thus the cost is bounded by $O(MD_{max})$ where D_{max} is the maximum vertex degree.

3.5.2 Directional distinct markers

If the N distinct super-glue markers are all directional, then such an oracle also defines a global ordering on each visited vertex thus marking both (visited) vertices and edges (i.e., a 'full paint can'). With such an oracle, both 'place validation' and 'back-link validation' processes are avoided. Without the need for both validations, the above mapping approaches are simplified to search algorithms such as Depth-first search (DFS) and GREEDY, which have $O(M)$ cost. Most recent work in this paradigm either deal with evaluating and improving DFS and other existing search algorithms, or address more challenging problems, such as constrained exploration.

In evaluating the existing algorithms and developing new ones, [82] 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 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 labeled map of the graph with known starting point and the other ends of all edges incident on the currently visited node.) The paper first shows that two natural exploration heuristics, the GREEDY and the depth-first search (DFS) algorithm cannot achieve this efficiency, i.e., the penalty is not linear in the number of nodes N in the graph. These two algorithms give penalty of order of M, which may become quite large in case of dense graphs. This paper then gives an exploration algorithm with penalty linear in the number of nodes N (only). In both GREEDY and DFS, the robot uses unexplored edges as long as possible and, when a current node v is 'saturated', i.e., all edges in v have been explored, it uses a simple strategy to reach a 'free' node v' which is incident to yet unexplored edges. In the case of GREEDY, v' is the free node closest to v, while in the case of DFS, v' is the most recently visited free node. Both these choices can be potentially inefficient: the vision of GREEDY is too local, and DFS does not make sufficient use of the knowledge of the explored

subgraph. Specifically, GREEDY performs poorly for the class of graphs in which at many stages of the exploration there exist free visited nodes which are sufficiently close to the current node v to 'attract' the robot far away from a still unexplored part of the graph. In order to come back to this unexplored 'hole', the robot performs many penalty traversals, i.e., traversals of explored edges. The reason why DFS may be inefficient is, in some sense, opposite to that for the GREEDY strategy. Based on its history, DFS always chooses the most recently visited free node and relocates there, even when the free node is far away from the current node. Hence the robot postpones exploration of regions that are close to the current node.

[82] then gives an exploration algorithm which performs at most $3N$ penalty traversals for every connected graph G. One of the main features of the algorithm is that after saturating the current node, the robot relocates itself along a dynamically constructed tree T. Suppose that the robot starts at node r of graph G. At any stage of exploration, let H denote the known subgraph of G. T represents a tree in H rooted at r and interconnecting only saturated nodes. (T is a spanning tree of H if and only if $H = G$.) During the execution of the algorithm, T is extended by adding new nodes and new edges. As above, the robot uses unexplored edges as long as possible. The place where the algorithm differs from GREEDY or DFS is the choice of the free node to which the robot relocates after the current node is saturated. As opposed to the above simple heuristics the algorithm explores the graph in the order given by the dynamically constructed tree T. The idea is to bring the robot back to the node of T at which it interrupted the construction of the tree. In doing this the algorithm tries to control the number of traversals through already explored edges (i.e., penalty traversals). Following the structure of the dynamic tree reduces penalty traversals and prevents the robot from being distracted from systematic exploration by free nodes situated close to it, which is the case for GREEDY. At the same time it excludes "temporal" (rather than "geographic") heuristics, which cause inefficiency of DFS.

Under the same paint can model, later work by the same authors [83, 29] investigates the impact of the amount of *a priori* topographic information available to an exploration algorithm on its performance. In this work a robot has to explore an undirected connected graph by visiting all its nodes and traversing all edges. In [83] 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, i.e., a labeled map of the explored graph given as input and is supplied with a '*sense of direction*'. Having 'sense of direction' means that when arriving at any node v, the robot knows this label and knows the label of the other end of every edge incident to v. ('Sense of direction' is revisited in the discussions on the 'whiteboard' model below.) 2) have only an un-oriented map of the graph, that is, it knows an unlabelled isomorphic copy of the graph but 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 this varying amount of knowledge on the exploration performance using an approach similar to competitive

<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 3.4: Summary of results in [29].

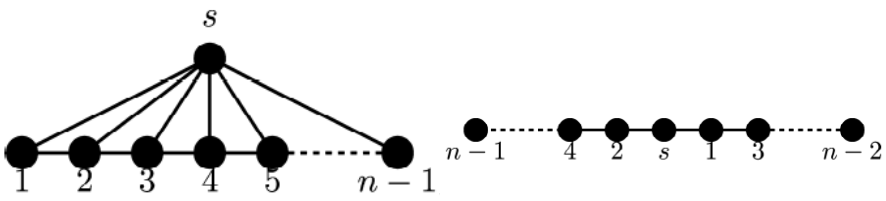
analysis, i.e., considering the ratio of the cost of an algorithm lacking some knowledge of the graph (maximized over all starting nodes) to that of the optimal algorithm having this knowledge (again maximized over all starting nodes). It is shown that the best exploration algorithm lacking any knowledge 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 [29] considers three similar but slightly different scenarios with different amounts of information. Under the first scenario, the robot does not have any *a priori* knowledge of the explored graph. Under the second scenario, the robot has an unlabeled isomorphic copy of the explored graph. This is called an *unanchored map* of the graph. Finally under the third scenario, the robot has an unlabeled isomorphic copy of the explored graph with a marked starting node. This is called an *anchored map* of the graph. Note that even the scenario with an anchored map does not give the robot any 'sense of direction', since the map is unlabeled (although the graph can be labeled). (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 measure in the above paper [83] as a measure of the quality of an exploration algorithm. Specifically, 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 shows that while for the class of all (undirected, 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 without any knowledge, DFS is still optimal for trees and lines but this is not the case if a map is available. Under the scenario with an unanchored map, the paper shows that for trees the optimal overhead is at least $\sqrt{3}$ but strictly below 2, and the optimal overhead for lines is $\sqrt{3}$ (and thus DFS, with overhead 2, is not optimal for trees and lines). Under the scenario with an anchored map, the authors construct optimal algorithms for trees and lines with the overhead of

3/2 and 7/5 respectively. A summary of the results is contained in Table 3.4. As an example, with the algorithm constructed for exploring a line with an unanchored map (i.e., the robot knows the length N of the line, but not its starting location) the robot starts by traversing in one direction for a distance of at most $\lfloor \frac{\sqrt{3}-1}{2} N \rfloor$. If it reaches an endpoint within this distance, it goes back to the other end and stops. If it does not reach an endpoint within this distance, then instead of continuing (as in DFS), it goes back to the other endpoint, and then goes to the other end and stops. This algorithm has a overhead of $\sqrt{3}$ and is shown to be optimal for exploring with an unanchored map.

Other work assumes a paint can oracle 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 [12, 5]. Later work [40] 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 (rope) 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 [12, 5] and tethered exploration [40], the world is modeled 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 it never confuses distinct locations. At any vertex the explorer can sense only the edges incident to it, and 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. As in [34] and the other work described above, the explorer only incurs a cost for traversing edges – thinking (computation) is free – and also a uniform cost is assumed for an edge traversal. 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. These work are reviewed below.

In piecemeal exploration [12, 5], the explorer is given an upper bound B on the number of steps it can make (edges it can traverse) in one exploration phase. In order to assure that the explorer can reach any vertex in the graph, do some exploration, and then get back to the start vertex, it is assumed that B allows for at least one round trip between starting location s and any other single vertex in G (i.e., the radius of G), and also allows for some number of exploration steps. Earlier work [12] first shows the failure or inefficiency of the simple approach that uses an ordinary search algorithm breadth-first search (BFS) or depth-first search (DFS), and the interrupt version of them (i.e., during exploration the robot just interrupts the search as needed to return to s, and then goes back to the vertex at which the search was interrupted and resumes exploration). For some environments DFS or interrupted DFS fails to reach all the vertices in the graph. BFS does guarantee that all vertices in the graph are ultimately visited but this may not be efficient enough for some environments. (Two example graphs illustrating failure



(a) A wheel graph with radius one. (b) A single path with s at the center.

Figure 3.9: Two example graphs illustrating failure of the DFS and BFS algorithms for constrained graph exploration. The starting node is labeled s, and the other nodes are labeled by their traversal order. (a) A Depth-First Search on this graph would fail for fuel tanks of size smaller than $2(N-1)$ or a rope of length less than N. (b) A Breadth-First Search is inefficient as it traverses $O(N^2)$ exploration steps.

of the algorithms for the piecemeal exploration is shown in Figure 3.9.) [12] refers to a search algorithm operating on a M edge graph as *efficiently interruptible* if it always knows a path back to s via explored edges of length at most the radius of the graph, and shows that a linear-time ($O(M)$) efficiently interruptible algorithm for searching an undirected graph can be transformed into a linear-time ($O(M)$) piecemeal search algorithm. Realizing that it seems difficult to piecemeal search arbitrary undirected graphs efficiently, the authors begin this line of research by focusing the attention on a special class of undirected planar graphs, called *city-block graphs*, which are grid graphs containing rectangular obstacles. For these graphs, the paper presents two efficient $O(M)$ search algorithms. Both algorithms are linear-time efficiently interruptible search algorithms (which can be transformed into a linear-time piecemeal search of a city-block graph). [12] shows that a robot can explore grid-graphs with rectangular obstacles in a piecemeal manner in linear time. In [5] the results were extended to show that the robot can learn *any* undirected graph piecemeal in *almost* linear time. [5] gives algorithms for piecemeal learning undirected graphs, and the most efficient (and complex) one gives an almost linear time algorithm which has a $O(M + N^{1+o(1)})$ running time. The piecemeal is most naturally satisfied by requiring the robot to explore in a near breadth-first manner. However, as shown in [12], the standard BFS is not always efficient for piecemeal learning. Let Δ denote the shortest-path distance from s to the vertex the robot is visiting, and let δ denote the shortest-path distance from s to the yet unvisited vertex nearest to s. The paper first shows that during piecemeal learning, a robot that maintains $\Delta \leq \delta$ (such as one using a traditional BFS) may traverse $O(M^2)$ edges. This paper thus solves the piecemeal learning problem by maintaining the *approximate* BFS constraint that the robot is never more than twice as far from s as is the nearest unvisited vertex from s, i.e. $\Delta \leq 2\delta$. The paper introduces the procedure Local-BFS which is a version of the traditional BFS

procedure that has been modified such that when expanding vertices on layer (breadth) i, the robot only explores vertices within a given distance-bound L of the given start vertex s. Using Local-BFS as subroutine, the paper introduces the Strip-Explore algorithm in which the robot explores the graph in strips of width L. The robot continues, strip by strip, until the entire graph is explored. By maintaining the approximate BFS constraint that $\Delta \leq 2\delta$, the algorithm has nearly linear running time. The work also poses the open problems of finding a linear-cost algorithm for general graphs. The piecemeal exploration algorithms in [5] is extended in later work [6] using the notion of sparse neighborhood covers [4], which gives $O(M + N \log^2 N)$ running time.

[40] settled the open problem posed in [5] by presenting a linear-cost algorithm for fuel constrained (piecemeal) and tethered robot exploration that runs with cost $\Theta(M)$. The paper shows that G can be explored by a tethered robot with cost $\Theta(M + N/\alpha)$ using a rope of length $(1 + \alpha)r$, where $\alpha > 0$ and r is the radius of G. By showing that the piecemeal learning model and the tethered learning model are equivalent within a constant factor (i.e., the two problems can be reduced to each other and the cost are equivalent within constant factors), the paper shows that the $\Theta(M + N/\alpha)$ tethered algorithm implies an $O(M/\beta)$ bound for fuel-constrained (piecemeal) exploration with a fuel tank of size $2(1 + \beta)r$, for any $\beta > \alpha$. The bounds hold regardless of whether r is known in advance. The paper first reiterates that neither the fuel-constrained search problem nor the tethered robot search problem can be solved using the classical search algorithms such as BFS and DFS (Figure 3.9.). The paper presents a bounded depth-first exploration bDFX algorithm, which is a recursive algorithm simply starts at a node v with a given rope length l and recursively visits all unvisited edges incident to v. The terminal condition is that no edges are visited if the length is 0. At the end of the call the robot is back to its original position v with the same initial rope length. In order to completely explore the graph, the bDFX algorithm is enhanced to run in phases until all vertices and edges are explored. At each phase of the enhanced algorithm, a set $T = \{T_1, T_2, ..., T_k\}$ is maintained which consists of vertex disjoint subtrees of G^* whose union contains all currently 'incomplete' vertices having unexplored edge(s). (At any stage $G^* \subseteq G$ is the subgraph consisting of all visited edges and nodes, those labelled explored or incomplete.) Initially, this set consists of one subtree containing only s. For each tree $T_i \in T$, the algorithm focuses on (one of) the node(s) $s_i \in T_i$ that is closest to s in the graph G^* (in terms of the number of edges to s). For each phase, the algorithm chooses the closest $s_i \in T_i$, sends the robot to s_i, and attempts to explore the incomplete nodes in T_i by doing a bDFX on each incomplete node. This process in turn creates new trees of incomplete nodes that are then added to T. However, simply allowing the robot to explore the entire subtree T_i would lead to an inefficient algorithm, particularly, when the robot attempts to reach an incomplete node of T_i that is deep in the search tree. The approach thus works by pruning T_i so that it has bounded depth. Pruning is performed by breaking T_i into smaller subtrees $T_{i_0}, T_{i_1}, ..., T_{i_k}$, and replacing these subtrees for T_i in T. The actual exploration is performed on the pruned tree containing the original closest node s_i. The process is repeated until there are no remaining incomplete trees.

The algorithm is shown to have cost linear in the number of edges M. The paper also proposes algorithms in which the radius of the graph is not known *a priori*, and algorithms for weighted graphs in which each edge has an associated length (weight).

3.5.3 Exploration on directed graphs

There are additional results on exploring directed graphs using N directional distinct super-glue markers (i.e., a full paint can oracle). Work in this paradigm includes [28, 2, 71, 48]. In the work the goal of the robot is to map an unknown directed, strongly connected graph by exploring all edges of the graph.

Deng et al. [28] showed that the Eulerian property is central in this problem. The main contribution is that they demonstrate 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. Realizing that there is no competitive strategy for graphs with unbounded deficiencies, [28] begins by looking at graphs with bounded deficiency. The paper presents an algorithm that achieves a bounded competitive ratio when the deficiency is bounded. (See the earlier discussion of the concept 'competitive ratio'.) It shows that if the graphs have deficiency one and the deficiency is known *a priori*, there is 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 [2] gave a first improvement to the algorithm in [28]. 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 show that this bound is tight for their algorithm by showing a matching lower bound of $d^{\Omega(\log(d))}M$. The proposed algorithm uses a divide-and-conquer approach. The robot explores a graph with deficiency d by exploring d^2 subgraphs with deficiencies $d/2$ each and uses the same approach recursively on each of the subgraphs. To create subgraphs with small deficiencies, the robot keeps track of visited nodes that have more visited outgoing edges than visited incoming edges, which are considered expensive because the robot, when exploring new edges, can get stuck there. The relocation strategy tries to keep portions of the explored subgraphs "balanced" with respect to their expensive nodes. The authors also gave lower bounds of $2^{\Omega(d)}M$ edge traversals for several natural exploration algorithms as Greedy, Depth-First, and Breadth-First.

Since the above work, there have been additional results in determining whether a graph with deficiency d can be explored by traversing $O(poly(d)M)$ edges. [71] develops a depth-first search algorithm that obtains the bound when the graph is dense, say, $M = \Omega(N^2)$. More recent work [48] gives the first generalized polynomial d algorithm. The main idea of the algorithm is to finish chains (i.e., newly discovered paths) in a breadth-first-search manner, and to recursively split off sub-problems that can be dealt

with independently because they contain new cycles which can be used to relocate without using other edges. The paper proves that the algorithm needs at most $O(d^8M)$ edge traversals. The authors conjecture that the bound can probably be improved.

The algorithms presented in [28, 2, 71, 48] are fairly long and complicated. Interested readers are encouraged to review the literature for algorithm details.

3.5.4 Summary

An oracle composed of N undirectional distinct super-glue markers answers 'exactly which vertex the currently visited vertex refers to' (i.e., a vertex paint can). Exploring undirected graphs with such an oracle provides movement-free 'vertex validation'. The cost is $O(MD_{max})$ there D_{max} is the maximum vertex degree. An oracle composed of N directional distinct super-glue markers also defines a global ordering on each visited vertex thus marking both (visited) vertices and edges (i.e., a 'full paint can'). With such an oracle both 'vertex validation' and 'back-link validation' are movement-free. Given the free validations, the above mapping approaches are simplified to search algorithms such as Depth-first search (DFS) and GREEDY, which have $O(M)$ cost. Work on exploring an undirected graph using a full paint can oracle either deal with evaluating and improving traditional search algorithms such as DFS, or address constrained exploration such as piecemeal and tethered exploration. The reviewed approaches for undirected graph have $O(M)$ cost.

A full paint can oracle is a sufficiently strong oracle, and has been investigated for exploration on directed graphs as well, and the best algorithm achieves cost of $O(ploy(d)M)$ where d denotes the *deficiency* of the graph, which is the minimum number of edges that must be added to make a graph Eulerian.

The solvability and the known cost of exploring with a paint can oracle are summarized in Table 3.5, together with that for the single marker and footprints oracles discussed in the previous sections. Several open questions of the topological mapping problem using different oracles are identified in the table, including exploration on directed graphs using different oracles. One another open question that has not been fully addressed in the literature is the power of multiple distinct *movable* markers. Dudek et al. show in [35] that using $1 < k < N$ multiple distinct movable markers on undirected graphs produces reduced cost over the single movable marker case. With multiple markers, rather than just searching a single unexplored edge, up to k unexplored edges are explored and the unknown ends are marked with unique markers. Once the k markers have been dropped, the entire explored sub-graph S is searched for the markers. As each marker is found, the search process given in the single robot algorithm [34] is used to incorporate this new information into the known subgraph. When the entire known sub-graph has been searched, all unseen markers are recovered, adding at most k new vertices to the sub-graph. [10] presents a deterministic algorithm that map a *directed* graph in polynomial time using $\Omega(\log \log N)$ distinct movable pebbles.

Here we present an empirical comparison of the performance of mapping with various

Oracle		Underlying graph-like world	
		undirected graph	directed graph
A super-glue marker	undirectional	no	no
	directional	yes. $O(M^2N)$	probably no
A movable marker	undirectional	yes. $O(MN)$	no
	directional	yes. $O(MN)$	not known
$M+N$ homogeneous markers (footprints)	undirectional	yes. $O(MN)$	not known
	directional	yes. $O(MN)$	not known
N distinct markers (a paint can)	undirectional	yes. $O(MD_{max})$	not known
	directional	yes. $O(M)$	yes. *O(poly(d)M)*

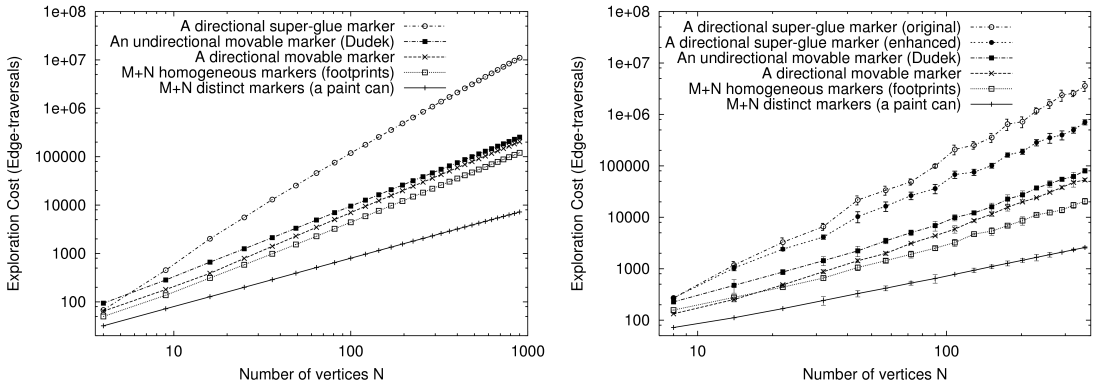
Table 3.5: Solvability and known cost of topological exploration with different oracle. Note that the trivial lower bound for the topological exploration and mapping problem is $M = |E|$.

oracles discussed so far. These include a directional super-glue marker, an undirectional movable marker, a directional movable marker, $M + N$ homogeneous markers (foot-prints), and $M + N$ distinct markers (paint can). Experiments are performed on both homogeneous and non-homogeneous lattice graphs. Results are provided in Figure 3.10. For these environments exploring with a movable marker (undirectional or directional), which has $O(MN)$ cost, produces cost reduction over exploring with a directional super-glue marker, which has $O(M^2N)$ cost. Exploring with foot-prints, which also has $O(MN)$ cost, provides a further cost reduction by a constant factor. Exploring with a paint can, which has a (optimal) $O(M)$ cost, provides the lowest cost over all the other oracles. In a non-homogeneous environment the enhanced super-glue pebble algorithm which also considers degree sequence produces reduced cost over the original version of the algorithm which only considers the door sequence $v_{k'}, ..., v_0$.

The next section briefly introduces some work using more powerful oracles, such as a *whiteboard* that can have information written on it. With such 'super power' oracles, exploration algorithms generally aim at addressing problems that are more challenging than the general exploration problem discussed so far. For example, to explore on dangerous environments.

3.6 Exploring with more powerful oracles

It is worth mentioning that in the topological exploration literature there exists work that assumes oracles that are more powerful than a paint can. For example, some work augments the robot with a movable marker that can have messages left on it, and some work associates each vertex with a *whiteboard* which the robot writes and reads messages there (i.e., an 'enhanced paint can' that can paint not only labels, but also other information). Both fixed and movable whiteboards are employed in the literature. Some work



(a) Homogeneous lattice.

(b) Lattice with 10% holes (missing vertices).

Figure 3.10: Performance of mapping with different oracles (log scale). Result for (b) is averaged over 30 graphs. Each graph has randomly generated holes (deleted vertices), and error bars show standard errors.

combines more than one such oracle. Usually, a more powerful oracle is used to address more challenging problems, which usually involve multiple robots. Some of this work is briefly introduced here.

[22, 9] address the distributed version of the graph exploration problem. In the problem 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, which should be consistent with one another in terms of the node labelling, i.e., the label assigned to any node should be the same in every robot's map. Given that the exploration is conducted in a distributed fashion, the problem is challenging and a more powerful oracle is employed. In [22], the whiteboard associated with each vertex is used both for marking the vertex, and more importantly, for providing 'indirect' communication among the robots. (Note that in the pebble-based multiple robot algorithm described earlier it is assumed 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. For example, during merging, the robot that wants its map to be merged (after comparing some messages on the whiteboards) writes "ADD-ME" and other information at the whiteboards of the places that it has explored. In this work either the number of robots k or the number of vertex N is known *a priori*. Under the same model, [9] addresses the more challenging case in which neither k nor N is known *a priori*, which is shown in [22] to be unsolvable using whiteboards as an oracle only. The oracle is strengthened by assuming that the edges on each vertex have labels and such labels provide a 'sense of direction' at the vertex. Sense of direction is a property of edge-labeled graphs [50]. Roughly speaking, having a sense of

direction is the simple ability 1) to tell, by looking at a sequence of labels corresponding to different paths starting from the same node, whether they end up in the same node or not, 2) to translate from neighbor to neighbor the encoded information about paths in the system. Using both the whiteboards and 'sense of direction' as oracle, the above problem becomes solvable without *a priori* knowledge of either N or k.

Other work including [49, 20] uses whiteboards combined with other oracles 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 the danger, multiple (distinct) robots are usually required to ensure that at least one will survive and finish the task. (The paper assumes the number of robots is greater than the known number of black holes and black links.) The goal is to construct a map of all the safe nodes and edges and have all the black holes and black links indicated. The problem is considered solved if at least one agent survives, and all surviving agents within finite time terminate having such a map. [49] uses as an oracle with both fixed whiteboards and movable whiteboards. (The movable whiteboards are termed 'markers' in the paper but the markers can have messages written on them). The fixed whiteboards are associated with each vertex, and the movable whiteboards are initially associated with each starting location of the robots but can be carried by the robots. One key technique in such work is that the robot explores an unexplored edge using a 'cautious walk', which is a way of labeling edges (doors) that protects subsequent agents from being eliminated by a black hole or black link. An edge is initially unexplored and has no label. When a robot leaves a node via an unexplored edge (door), the door is marked 'dangerous' on the associated whiteboard. (The message will be updated to 'explored' if the robot come back safely.) Agents are not allowed to enter an edge (door) marked 'dangerous'. Since black holes and black links eliminate agents, the first agent entering an edge that leads to a black hole or black link will mark it 'dangerous' (permanently) and prevent all the other agents from being eliminated by entering the same port. Movable whiteboards contain partial maps and other information and are used in the merging process of the algorithm.

3.7 Exploring with an impoverished oracle

A graph-like world can be fully explored and mapped using additional aids such as a pebble (marker), footprints, or a paint can, even if there are no spatial metrics and little sensory ability on the part of the robot. Using an appropriate oracle, algorithms can be developed which are provably correct (deterministic) solutions to the SLAM problem. The next chapter is devoted to oracle-less approaches to SLAM. Without an oracle, additional information is needed. The choice of the approaches described in the next chapter is the use of metric information. Somewhere in between the two paradigms lies the interesting work by Dudek et al. [32, 33] which proposes *oracle-less* approaches *without* any additional information, and [109] which addresses exploring with *insufficient* oracles. The challenges

faced by the approaches signify the importance of using (sufficient) oracles or otherwise using additional (metric) information. These are described below.

3.7.1 Exploring without an oracle

Under the same world and robot sensing model as assumed in [34, 36], Dudek et al. [32, 33] investigate the possibility of exploring the same world model *without* such a oracle, even though individual locations may not be uniquely identifiable. The paper presents an exploration approach simply based on the structure of the world itself. In contrast to the above oracle-based approaches such as [34] where a single unique solution is produced, here multiple models of the unknown world may result. In the oracle-less approach, while the exploration takes place, the robot constructs a data structure called the "*exploration tree*" which includes, at the end of the exploration, the set S of all possible world models (i.e. maps) consistent with the robot's observations. This set of solutions is called the "solution universe". The exploration tree refers to the collection of possible partial maps that serve as hypotheses about the world the robot is exploring. It is constructed incrementally while the exploration takes place. The *root* of the tree is a map containing only the initial place from which the exploration began. A *level* in the tree corresponds to the traversal of a previously unexplored edge. The *nodes* belonging to a given level 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.

As in previous marker-based exploration problems [34], in [32, 33] one of the critical questions to solve is the 'have I been here before' problem, which is challenging here 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 defined 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 existing nodes); Thus, an erroneous edge is added to the world.

Branches in the exploration tree are created as a result of modeling 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

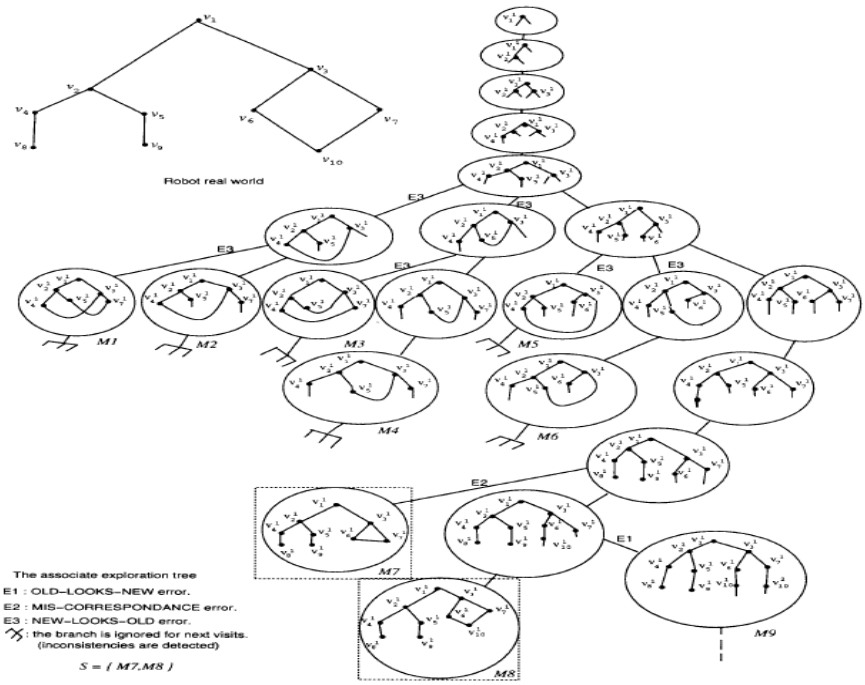


Figure 3.11: Exploration tree and three types of errors. Courtesy of Dudek et al. [32].

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, i.e., a leaf which faithfully describes the true connectivity in the world. (See Figure 3.11 for an example of exploration tree and different types of errors. In the example solution universe S consists of leaf node $M7$ and $M8$, with $M8$ being the correct model.) Typical exploration trees usually include branches that are subsequently pruned (i.e. they develop inconsistencies before they lead to a complete model). The major reason for this is the weakness of the signature information used by the robot in addressing the place identification problem. To make the exploration more robust and effective, the algorithm exploits non-local information by defining an *extended signature* incorporating signature information about a place's neighbors. The idea is that while the signature (i.e., degree) of any single place may not be unique, under appropriate conditions the distinctiveness of a particular set of signatures in a neighborhood increases with neighborhood size.

The worst case behaviour of the algorithm is clearly problematic. Despite the availability of an extended signature, ambiguity may still remain 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 the single cycle graphs shown in Figure 3.1 every place is identical to every other and the number of possible models grows infinitely,

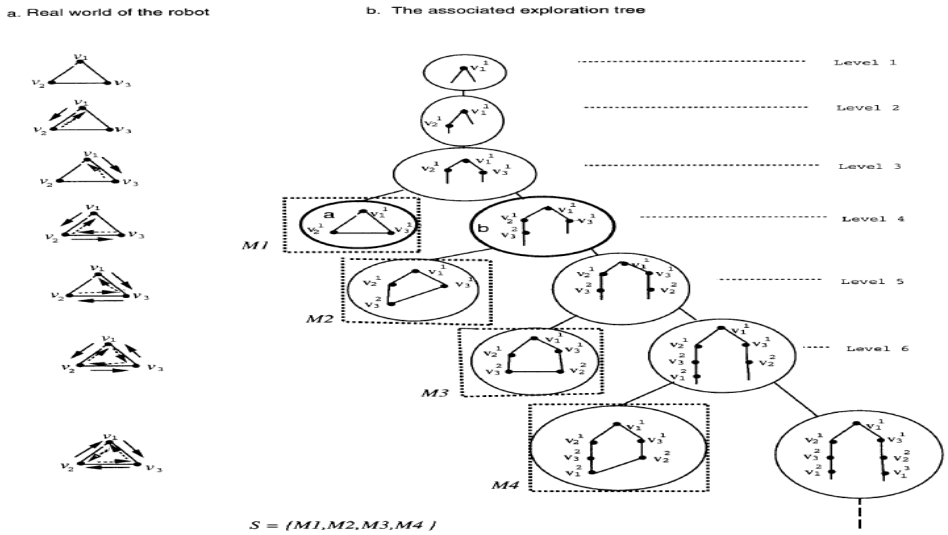


Figure 3.12: Problems with constructing the exploration tree for a regular graph. Courtesy of Dudek et al. [32].

as shown in Figure 3.12. This difficulty is not surprising since under such circumstances the algorithm is attempting to construct a map from no knowledge about where the robot is or how the robot is moving. The paper also suggests some additional ways of reliably exploring unknown worlds when both the extended signature and the existence of a uniquely distinguishable place are not sufficient. For example, the total number of places, information about the probability distribution of place signatures, or planarity of the world being explored.

The marker-less approach in [32, 33] was revisited by Dudek et al. in recent work [39]. The paper presents a new exploration strategy called *breadth-first ears traversal* (BFET) that can be used in planar graph. Essentially BFET works by eliminating inconsistent models through the 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 then describes a beam-style search algorithm which bounds the number of hypotheses maintained at each step of the exploration process based on heuristic evaluation function. Following Occam's razor principle, the paper assumes that the simplest models capable of explaining the observed data are the best ones and rank them accordingly. Accordingly, the paper defines a simple hypothesis as one with as few vertices as possible. 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. The top N of these models are then selected for maintenance and the rest are discarded. This approach can be considered conceptually similar to a particle filter (discussed in the next chapter). The algorithm shows improved performance in some environments

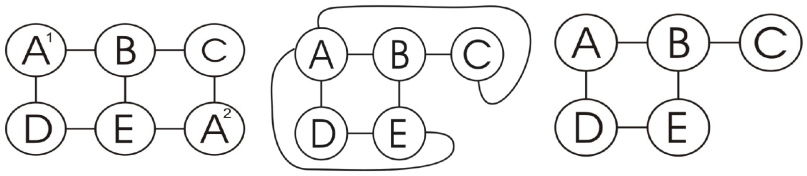
over the previous work [33]. Graph size, sparseness of graphs are factors influencing the performance of the new algorithms.

3.7.2 Exploring with insufficient oracles

In contrast to the above work that assumes no oracle, some other interesting work assumes a limited oracle with which some (but not all) vertices of the graph can be uniquely identified but the others are ambiguous. One such work [109] 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 modeled as a labelled graph. The vertex labels of the graph represent deterministic (but potentially aliased) sensor readings that characterize the places. If the set of labels is smaller than the set of vertices, several vertices share the same label. We can think this as providing the robot with a paint can and a set of labels, but in contrast to the assumptions that the set of labels can uniquely identify all vertices, here the set of labels may be smaller than the number of vertices (so the robot has to paint duplicated labels on some of the vertices).

The paper proposes an approach to infer a topological map from sequences of (repeated) vertex labels obtained from the traversal of the environment. The proposed approach bears some similarities to the oracle-less approach by Dudek et al. [33, 39]: 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) Occam's razor principle is used to construct a small map in terms of vertices that best explains the sequence of labels.

The environment graph is first traversed by the robot, who has no knowledge of the environment. The robot traverses the environment extensively, first 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 3.13(a) and also assume that during subsequent traversal the observed labels 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)* = {(A,B), (B,C), (C,A), (A,E), (E,D), (D,A), (B,E), (E,A)} and *Grams(h,3)* includes 3-grams (A,B,C), (B,C,A), (C,A,E), etc. The proposed algorithm infers the map by merging the n-grams using a stochastic local search with respect to the mapping constraints. The mapping constraints include *hard* constraint and *soft* constraint. The *hard* constraint is that the neighborhood of each vertex of the environment graph corresponds to the neighborhood information in the map graph. That is, the inferred map graph must exclusively explain



(a) (b) (c)

Figure 3.13: (a) Example environment graph. Also a good hypothesis map. Note the two aliases, labelled A^1 and A^2. (b) An incorrect hypothesis map. (c) Possible map G_{map} to be merged. Courtesy of [109].

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 the process 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 unsuccessful if it violates the mapping constraints. 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 merged is arbitrary. Usually for a n-gram γ there are several possibilities for merging γ to G_{map}, resulting in different inferred map graphs. To test whether a merge possibility is appropriate, a set M of all merge possibilities to merge γ with G_{map} is calculated and sorted in ascending order according to the number of vertices the resulting map graph requires. Then, beginning with a merge possibility that requires the fewest vertices, every merging possibility $m \in M$ is tested to see whether it satisfies the (hard) mapping constraints. If a merge is unsuccessful, it is removed from M and the next merging possibility in M is tested; If a merge possibility is successful, all other merge possibilities which require more vertices are immediately removed from M due to the soft constraint. To test whether a merge possibility m satisfies the hard constraint, a 'hypothesis map' is generated showing the result of the potential merging. Then, a set of local n-grams is extracted from the hypothesis map (by virtually 'traversing' the hypothesis map), and is then compared with $Grams(h, n)$. An n-gram that is contained in the set of local n-grams but not in *Grams(h,n)* indicates a violation of the consistency constraint and results in the potential merge being abandoned. For example, there are two possibilities to merge the 3-gram (C,A,E) with the G_{map} shown in Figure 3.13(c), resulting in the hypothesis maps shown in Figure 3.13(a) and 3.13(b). The potential merging that requires fewer vertices is tested first (Figure 3.13(b)). However,

this potential merge violates the mapping constraints and thus must be abandoned: the hypothesis map contains some local n-gram (e.g.,(E,A,D) and (C,A,D)) that is not in $Gram(h, n)$. 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. 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. Experiments investigate whether the environment graph is isomorphic to the inferred map graph as a measure of the quality of the maps 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 alising exceeds a certain limit (e.g., for label set cardinality of no more than 60%), then using 5-grams generated about 50%–70% isomorphic maps, whereas using 3-grams typically generates less than 50% isomorphic maps.

3.8 Summary

This chapter reviewed notable work on robotic exploration with different oracles. In such work the robot has weak sensing abilities so they cannot distinguish locations alone. Therefore an 'oracle' that helps the robot solve the 'have I visited here before' problem is needed. The oracle-based SLAM problem is typically formulated in terms of a topological representation of space, and has been formulated for both directed and undirected spaces and spatial representations. Given an embedded graph, it is not, in general, possible to map the graph without resorting to the use of some type of oracle. Given an oracle of sufficient power (e.g., a directional super-glue pebble), mapping is possible with cost $O(M^2N) \leq O(N^5)$. The expected cost bound for mapping is $O(M)$ and this can be obtained with a sufficiently powerful oracle such as a paint can. (See Table 3.5 in section 3.5 for a summary.) Algorithms have been developed for related problems including 'more powerful oracles' that provide more information than is necessary to obtain $O(M)$ mapping, and weaker oracles that only provide probabilistic mapping. The reviewed work is summarized in Table 3.6.

In addition to the oracles discussed in this chapter and the open problems associated with them (e.g., using the oracles on diected graphs), there are some other interesting oracles that are not fully addressed in the literature. One such an example is a string. A string oracle is an interesting alternative to a marker-based oracle in that there are many ways of manipulating the string, resulting in oracles of different powers. Here we describe some string examples that can be mapped onto the various oracles discussed above. Unlike a marker which can only be placed at a particular location, a string can be tied to a particular vertex and then laid out through the graph. A string therefore marks a vertex as well as edges in various ways. If the string is short (e.g., no longer than one

Summary of oracle-based approaches		
Oracle model	Undirected graph	Directed graph
A super-glue marker	Dudek et al. [34, 37] Hui et al.	
A movable marker	Dudek et al. [34, 36, 38, 37] Hui et al. [105, 106, 107] – (*multi-robot*) Rekleitis et al. [91] – (*planar graph*) Deng et al. [27, 25] – (*edge marker*[25])	Bender & Slonim [11] Bender et al. [10]
Footprints	Deng et al. [26, 27]	
A paint Can	Dudek et al. [35] Panaite & Pelc [82] Pelc et al. [83, 29] Awerbuch et al. [12, 5, 6] – (*piecemeal*) Duncan et al. [40] – (*tethered*)	Deng et al. [28] Albers & Henzinger [2] Kwek [71] Fleischer & Trippen [48]
More powerful Oracles	[22, 9] – (*whiteboards etc*) [49, 20] – (*movable whiteboards etc*)	
No oracle	Dudek et al. [32, 33, 39]	
Insufficient oracle	Werner et al. [109]	

Table 3.6: Exploration algorithms reviewed in this chapter.

edge length), then the robot can tie it at the starting node v_0 and lay it out toward some direction, using it as a directional super-glue marker; The robot can also carry the short string during exploration, using it as a (directional) movable marker by tying it at each new location. If the string length l is no shorter than the total length of all edges, i.e., the string is at least as long as the graph size ($l \geq |E|$) (assuming a unit edge length), then the string can be used the same way as homogeneous markers (i.e., toeless footprints). In this case if the string itself contains direction marks on it (the surface of the string provides specific direction information), then it acts as directional homogeneous markers (i.e., footprints with toes). If the string is much longer than the graph size ($l \gg |E|$), then the robot can also tie distinct knots at each visited location, using the string as distinct markers (a vertex paint can). In this case if the string itself or the knots contain direction information (e.g., each knot is tied near a specific entry edge, or an extra knot is tied along a specific entry edge), then the string acts as directional distinct markers (i.e., a full paint can), which incurs $O(M)$ optimal cost.

4 Exploring with metric information

The previous chapter reviewed robotic exploration with oracles. Exploration using oracles is well studied and, depending on the 'power' of oracle, exploration cost ranges from $O(M) \leq O(N^2)$ to $O(M^2N) \leq O(N^5)$. Note that $O(M)$ is the cost bound that can be obtained with a paint can and $O(M^2)$ is the cost bound of directional super-glue – the weakest oracle discussed in the last chapter, and perhaps the weakest of all possible oracles. With the help of an oracle, deterministic solutions to the SLAM problem exist. Although an oracle is helpful, manipulating oracles is expensive in general. Moreover, specific oracles might not always be available. At the end of the last chapter we review the oracle-less work by Dudek et al. which demonstrated that simply depriving the robot of any oracle (without adding other information) will not generate a unique world model, even though, in the model perfect sensing and motion are assumed. For example, for a simple three node cycle, the oracle-less algorithm will generate an infinite number of models.

Without an oracle, we thus have to resort to other knowledge about the world. That is, we need some information describing the underlying environment. One obvious choice is metric information. Metric-based approaches can be considered as associating pose (metric) information with each node and edge in the topological representation.

4.1 Exploring with perfect metric data

Before introducing the metric representation and probabilistic framework that are generally adopted in the literature, it is worth mentioning first that if the robot has a perfect metric sensor, i.e., each location is associated with prefect pose information, then the mapping problem is trivially solvable. Having perfect sensing (thus metric pose information) is equivalent to having a 'paint can' with which the explorer can distinguish each location. This assumption, however, is not realistic. In reality, sensors are not perfect and therefore errors may exist in the motion model, the measurement model or both (including signature perception). Therefore it is more realistic to assume noisy metric information.

4.2 Exploring with noisy metric data

Exploring with noisy metric data typically uses metric representation and resorts to probabilistic framework.

Metric spatial representation Work in this paradigm uses metric (pose) information associated with robot motion to obtain metric spatial representations of the underlying world. Metric maps adopt a metric representation of space, which captures the metric properties (e.g., coordinates in a Cartesian representation) of the environment. A representative example of a metric map is the occupancy grid map [44], in which the space is represented as a fine-grained grid defined over the continuous space of locations. Compared with topological representations that are concise, metric representations provide a more detailed world representation. For example, an occupancy grid map can be used to represent unstructured environments. The disadvantages of metric maps include the large storage requirements associated with them [98].

Probabilistic framework In the last chapter we saw that by exploring with an oracle, the SLAM problem can be solved deterministically. Without an oracle, and with imperfect sensing and plant (noise) models, however, the dominant approach is to use probabilistic concepts to explicitly represent and manipulate spatial uncertainty. Early work with probabilistic representations includes [41, 43, 17, 77, 79]. Since the 1990's, with the application of powerful statistical frameworks for simultaneously solving the mapping problem and the induced problem of localizing the robot relative to its growing map [94, 95], the field of metric robot mapping has been dominated by probabilistic techniques. Probabilistic algorithms approach the problem by explicitly encoding (modeling) these different sources of noise and their effects on the measurements and the map models. Under a probabilistic framework, the disambiguation tasks becomes the process of estimating the likelihood of different locations under the associated probabilistic functions.

In robotic mapping and localization the task is usually modeled as inferring a state quantity x (map or robot pose), based on some data d (measurement or control). Within a probabilistic framework, this is represented as the conditional probability $p(x|d)$, often referred to as a *posterior probability*, or *belief* [98]. When mapping and localization were introduced by researchers in the early 1980's, the work focused on solving the two problems of mapping and localization independently. We first briefly mention the probabilistic framework for the two problems.

Localization without mapping Within a probabilistic framework, localization without mapping can be expressed as estimating the posterior probability $p(s_t|z^t, u^t, s^{t-1}, \Theta)$, where s_t is the robot pose (position) at time t, $z^t = \{z_1, z_2, ..., z_t\}$ is sensor reading (history) up to time t and $u^t = \{u_1, u_2, ..., u_t\}$ is the control (history) up to time t, $s^{t-1} = \{s_1, s_2, ..., s_{t-1}\}$ is the path (set of poses of the robots up to and including time $t-1$) and Θ is the known map, usually assumed to be static (so subscript t is omitted). Solutions to the localization problem usually use estimation techniques such as the Extended Kalman Filter (EKF) or particle filters to estimate the posterior [72, 18, 100]. (These two techniques are discussed briefly later in this chapter.)

Mapping without localization Metric maps such as occupancy grids are commonly used for this mapping (with pose) problem [45, 79]. In this approach, the world is represented as a fine grained grid where each cell is a random variable, corresponding to the occupancy of the location it covers. Occupancy grid mapping algorithms implement approximate posterior estimation for these random variables. Specifically, the approach calculates the posterior over maps given the data, i.e., $p(\Theta|z^t, s^t)$, where Θ is the map, z^t is the set of all measurements up to time t, s^t is the path of the robot defined through the sequence of poses occupied by the robot. Sensor readings are converted into a probability distribution using a stochastic sensor model [99].

Joint estimation (SLAM)

The discussion starts from the probabilistic framework of SLAM, followed by a brief overview of the metric SLAM approaches in the literatures.

State in SLAM and the Markov assumption In SLAM problem, the robot pose s and the map Θ together constitute the state x, i.e., $x_t = (s_t, \Theta)$. The robot's pose changes over time, while the map Θ is usually assumed to be *static*. Within the framework, a state $x_t = (s_t, \Theta)$ is assumed to be *complete*, meaning that the current state is a sufficient predictor of the future and therefore the past state and measurements carry no additional information to help predict the future more accurately. Temporal processes that meet these conditions are commonly known as *Markov chains* [67].

Probabilistic generative laws In probabilistic SLAM, the evolution of the state and measurements are governed by probabilistic laws. In general the state x_t (i.e., s_t and Θ) is generated stochastically from the state x_{t-1} (i.e., s_{t-1} and Θ) and therefore the emergence of x_t might be conditioned on all past states, measurements and controls. Hence the probabilistic law characterizing the evolution of state is given by a probability distribution $p(s_t, \Theta|s^{t-1}, \Theta, z^{t-1}, u^t)$. Given that the state is complete, and therefore is a sufficient summary of all that happened in previous time steps, x_{t-1} (specifically s_{t-1}) is a sufficient statistic of all previous controls and measurements up to time $t-1$ (i.e., z^{t-1}, u^{t-1}), i.e.,

$$p(s_t, \Theta|s^{t-1}, \Theta, z^{t-1}, u^t) = p(s_t|s_{t-1}, u_t).$$

The probability $p(s_t|s_{t-1}, u_t)$ is called *state transition probability* or *motion model* [99]. The state at time t is dependent only on the state at time $t-1$ and the control at time t. Likewise, according to the complete state (Markov) assumption, the process by which measurements are being generated can be modeled by

$$p(z_t|s^t, \Theta, z^{t-1}, u^t) = p(z_t|s_t, \Theta).$$

The probability $p(z_t|s_t, \Theta)$ is called the *measurement probability* or *perceptual model* [99]. The measurement at time t depends only on the state at time t. The state transition

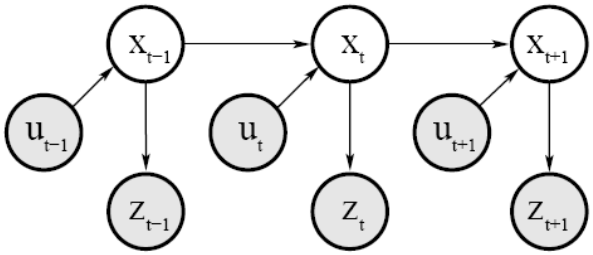


Figure 4.1: The dynamic Bayes network that characterizes the evolution of controls, states and measurements. Courtesy of S. Thrun [99].

probability and the measurement probability together describe the dynamic stochastic system of the robot and its environment. Figure 4.1 illustrates the evolution of states and measurements, defined through these probabilities. Such a temporal generative model is also known as *hidden Markov model* (HMM) or *Dynamic Bayes Network* (DBN) [99].

SLAM posterior A key concept in SLAM is that of a *belief*. A belief reflects the robot's internal knowledge about the state of the environment. Within a probabilistic framework, the solution to the SLAM problem is modeled as estimating the belief, which is represented by a posterior probability distribution over all possible states, i.e., all possible maps and all possible robot poses, given (conditioned on) the knowledge of the controls and measurements accumulated by the robot. This distribution is also called the *SLAM posterior*[98, 99, 78]:

$$bel(s_t, \Theta) = p(s_t, \Theta | z^t, u^t)$$

where the current pose s_t of the robot and the map Θ is conditioned on the set of all sensor readings z^t and controls u^t.†

Bayes Filter for SLAM posterior Following the Bayes' rule and using the motion model and the perceptual model, the SLAM posterior at time t can be computed recursively as function of the posterior at time $t-1$. This recursive update rule, known as the *Bayes filter* for SLAM, is the basis and the most general algorithm for calculating the SLAM posterior and is given by ([98, 99, 78])

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

†Sometimes it is assumed that the landmarks are uniquely identifiable. In this case it also takes as input the *data association* or *correspondence* $n^t = \{n_1, n_2, ..., n_t\}$ which represent the mapping between map points in Θ and observations in z^t. Then the belief becomes $p(s_t, \Theta | z^t, u^t, n^t)$.

The first term η is a normalizing value that ensures the posterior is a probability and is within the range of [0,1]. The Bayes filter extends Bayes' rule to temporal estimation problems. Represented using the notation of *belief (bel)*, the filter can be written more compactly as

$$bel(s_t, \Theta) = \eta \underbrace{p(z_t|s_t, \Theta)}_{\text{perceptual model}} \int \underbrace{p(s_t|s_{t-1}, u_t)}_{\text{motion model}} bel(s_{t-1}, \Theta) ds_{t-1}. \tag{4.1}$$

The evaluation of the filter possesses two steps. Evaluating the integral part that involves the prior belief over s_{t-1} and control u_t is often called *prediction* or *control update*, as it predicts the state at time t based on the previous state posterior (before incorporating the measurements). Multiplying the prediction by the perceptual model is known as *measurement update*. A full description and mathematical derivation of Bayes filter applied to SLAM can be found in [99] and [78].

Full SLAM versus online SLAM The above problem is called the *online SLAM problem* since it involves the estimation of robot state at time t, i.e., estimate $p(s_t, \Theta|z^t, u^t)$. From the probabilistic perspective, there is another form of problem called the *full SLAM problem*. In full SLAM, also called *offline* SLAM, the aim is to calculate a posterior over the entire path s^t along with the map ($s^t = s_1, s_2, ..., s_t$), instead of just the current pose s_t, i.e.

$$p(s^t, \Theta|z^t, u^t).$$

The online SLAM problem is the result of integrating out past poses from the full (offline) SLAM problem.

Estimating the posterior In general, the SLAM posterior in Equation 4.1 cannot be evaluated in closed form. Thus, mapping algorithms resort to additional assumptions in order to solve it. These assumptions and their implications on the resulting algorithm and maps constitute the primary differences between different solutions to the SLAM problem. Two popular families of estimation techniques are *Gaussian Filters*, and *Nonparametric Filters* [99]. In Gaussian techniques the posterior are represented by multivariate normal distributions. Important Gaussian filter algorithms include the *Kalman filter* and *extended Kalman filter* (or EKF) [66]. An alternative to Gaussian techniques are nonparametric filters. Instead of relying on a fixed functional form of the posterior, nonparametric filters approximate the posterior by a finite number of values, each roughly corresponding to a region of the state space. Some nonparametric filters rely on a decomposition of the state space, in which each such value corresponds to the cumulative probability of the posterior density in a compact subregion of the state space. An example of this technique is *Histogram filter* [99]. Others approximate the state space by random *samples* drawn from the posterior distribution. One important sample-base technique is the *particle filter* [99]. The following sections present brief introductions to

Kalman filters and particle filters.

(Extended) Kalman Filters and SLAM via EFK

The description for Kalman Filters and Extended Kalman Filters follows that given in [108], [99], [98] and [61]. Kalman filters, introduced in [66], are Bayes filters that represent the posterior with multivariate normal distributions (Gaussians), which are parametrized by two sets of parameters: the mean μ which describes the most likely state of the robot and landmarks, and the covariance Σ that encodes the pairwise correlations between all pairs of state variables. (μ is a vector that possesses the same dimensionality as the state, and the covariance Σ is a symmetric and positive-semidefinite matrix with dimension being the dimensionality of the state squared.)

In addition to the Markov assumptions of the Bayes filter, Kalman filters make three assumptions: 1) The motion model (state transition probability) is a linear function in its arguments with added Gaussian noise; 2) The perceptual model (measurement probability) is a linear function in its arguments, with added Gaussian noise; 3) the initial belief is known normally distributed. These three assumptions are sufficient to ensure that the posterior at time k, which is represented by the mean μ_k and the covariance Σ_k (moments parameterization), is always a Gaussian, for any point in time k. (Each of these assumptions is relaxed within the EKF framework described next.)

The motion model $p(x_k|u_k, x_{k-1})$ is a *linear* function in its arguments with added Gaussian noise. This is expressed by

$$x_k = A_k x_{k-1} + B_k u_k + w_k$$

where x_k and x_{k-1} are state vectors and u_k is the control vector at time k. A_k is the state transition model (matrix) which is applied to the previous state x_{k-1}; B_k is the control-input model (matrix) which is applied to the control vector u_k; w_k is a multivariate Gaussian random vector that models the uncertainty introduced by the state transition, called *process* or *plant noise*. Its mean is zero and its covariance is denoted Q_k.

The perception model $p(z_k|x_k)$ is also assumed to be linear in its arguments, with added Gaussian noise

$$z_k = H_k x_k + v_k$$

where H_k is the observation model which maps the true state space into the observed space. v_k is the *measurement noise* which is assumed to be multivariate Gaussian with zero mean and covariance R_k.

The state of the filter is represented by two variables:

- $\hat{x}_k$, the estimate of the state at time k;
- P_k, the error covariance matrix (a measure of the estimated accuracy of the state estimate) at time k.

As an implementation of the general Bayes filter, the Kalman filter possesses two distinct phases: **Prediction** and **Measurement Update**. The Prediction phase uses the estimate from the previous time step to produce an estimate of the current state. In the Measurement Update phase (also called the Correction phase), measurement information from the current time step is used to refine (correct) this prediction to arrive at a new, (hopefully) more accurate estimate.

Prediction phase The estimated *a priori* state $\hat{x}_k^-$ is formed based on the previous estimate of the state and the current value of the input

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k.$$

The *a priori* covariance, denoted P_k^-, is calculated as

$$P_k^- = AP_{k-1}A^T + Q$$

where A^T denotes the transpose of the matrix A. Note that these two equations use previous values of the *a posteriori* estimate of state $\hat{x}_{k-1}$ and covariance P_{k-1}.

Measurement Update phase To refine (correct) the *a priori* estimate, a Kalman filter gain K_k is used. K_k is defined as

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R}.$$

The gain is used to refine the *a priori* estimate to give the posterior estimates.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-).$$

The difference $(z_k - H\hat{x}_k^-)$ reflects the discrepancy between the predicted measurement $H\hat{x}_k^-$ and the actual measurement z_k, and is called the measurement *innovation*. The *a posteriori* covariance is also refined and is used in next time step:

$$P_k = (I - K_k H)P_k^-$$

where I denotes the identity matrix. The Kalman filter is optimal in a least-squares sense [96]. A pictorial illustration of the operation of the Kalman filter is shown in Figure 4.2.

Extended Kalman filter The basic Kalman filter is limited to linear models. However, most non-trivial systems are non-linear. The non-linearity can be associated either with the motion model or with the perceptual model or more generally with both. The *extended Kalman filter* or EKF [108, 99] relaxes the linearity assumption. Here the assumption is that the state transition probability and the measurement probabilities are governed by

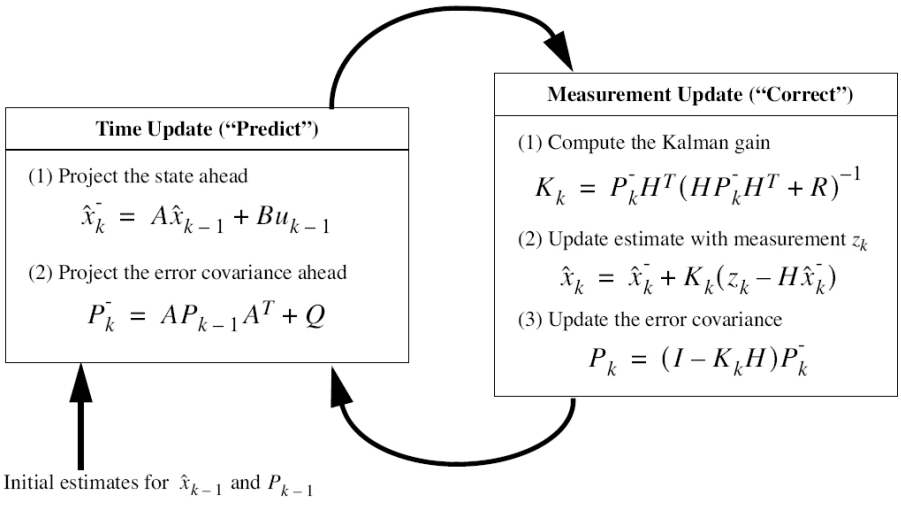


Figure 4.2: The operation of the Kalman filter. Courtesy of G. Welch [108].

nonlinear functions $f()$ and $h()$, respectively:

$$x_k = f(x_{k-1}, u_k) + w_k.$$
$$z_k = h(x_k) + v_k.$$

With arbitrary functions $f()$ and $h()$, the belief is no longer a Gaussian and performing the belief update exactly is usually impossible, and the Bayes filter does not possess a closed-form solution and thus the EKF must resort to an additional approximation. The key idea underlying the EKF approximation is to linearize the nonlinear functions (f and h) about the current state estimation. EKF utilizes a linearization method based on a (first order) *Taylor expansion*. Linearization approximates the nonlinear function by a linear function that is tangent to the function at the mean of the Gaussian. In most robotics problems, state transitions and measurements are nonlinear. The goodness of the linear approximation applied by EKF depends on two main factors: the degree of uncertainty and the degree of local non-linearity of the functions that are being approximated. The higher uncertainty and higher nonlinearities typically results in larger approximation errors. A full description and mathematical deviations of EKF can be found in [99].

Kalman filtering approaches to SLAM were introduced by Smith, Self and Cheeseman [94, 95], and first implemented in [80]. They estimated the spatial-relationships between landmarks using an EKF framework, in the context of feature-based mapping with landmarks. More recent work includes [19, 30, 73, 81]. In practice, EKF SLAM has been applied with great success. Although EKF SLAM has proven effective in many domains it has a number of concerns, including its sensitivity to linearity and quadratic

complexity.

SLAM via Particle Filters

The above Kalman filter and the EKF represent probability distributions using a parameterized model (a multivariate Gaussian). Particle filters [99, 53], on the other hand, represent distributions using a finite set of sample states, or "particles". Regions of high probability contain a high density of particles, whereas regions of low probability contain few or no particles. Rather than assuming a closed form representation of the SLAM posterior, the core concept of the nonparametric particle filter approach is to represent the belief $bel(x)$ by a set of m weighted samples distributed according to $bel(x)$: $bel(x) = \{x^{(i)}, p^{(i)}\}_{i=1,\ldots,m}$. Here each $x^{(i)}$ is a sample (a state), and the $p^{(i)}$ are non-negative numerical *importance factors*, such that $\sum_{i=1}^{m} p^{(i)} = 1$. As the name suggests, the importance factors represent the weight (importance) of each sample.

The recursive update is generally realized in the following steps [53], computing the expression in Equation 4.1 from right to the left.

For each sample particle x_{t-1} do:

1. Sample a state x_{t-1} from $bel(x_{t-1})$, by drawing a random $(x_{t-1}^{(i)})$ from the sample set representing $bel(x_{t-1})$ according to the (discrete) distribution defined through the importance factors $p_{t-1}^{(i)}$.

2. Use the sample $x_{t-1}^{(i)}$ and the control u_{t-1} to sample $x_t^{(j)}$ from the distribution $p(x_t|x_{t-1}, u_{t-1})$. The predictive density of $x_t^{(j)}$ is now given by the product of $p(x_t|x_{t-1}, u_{t-1})$ and $bel(x_{t-1})$.

3. Compute the new weight for the particle, accomplished by computing the the likelihood of the sample $x_t^{(j)}$ given the measurement z_t. Weight the sample $x_t^{(j)}$ by the (non-normalized) new importance factor $p(z_t|x_t^{(j)})$.

Then after the generation of m samples, the new importance factors are re-normalized. (This procedure implements the normalizer η in Equation 4.1.) After this step the particles represent a probability density function, implementing the posterior in 4.1. Finally, re-sampling of the posterior is performed. The goal is to sample the posterior so that only the most important particles remain in the set for the next iteration. There are many ways to implement the resampling stage but a widely used method is described by Liu in [76], which notes that resampling is necessary for several reasons, including to prune away 'bad' samples, and produce multiple copies of 'good' samples which allow the next sampling stage to produce better future samples.

Particle filters have become an attractive alternative solution to the SLAM problem. Notable Particle filter approaches to SLAM includes DP-SLAM [46], a 'pure' particle filtering approach to SLAM, which uses a particle filter to maintain a joint probability

density over robot positions as well as the possible map configurations. FastSLAM [78] is another notable algorithm that uses modified particle filter to solve the SLAM problem.

Metric SLAM approaches

Many SLAM approaches involve implementation and deployment of SLAM algorithms in large scale environments. There are many dimensions along which the SLAM approaches can be categorized. For example, they can be categorized based on the way the map representation is maintained: some of the methods employ a single, globally referenced coordinate frame for state estimation. Some do not maintain a single, global coordinate frame, but rather generate a short-term submap with its own local coordinate frame. Alternatively the approaches can be categorized into three broad categories, based on the basic principles as well as the optimality that is maintained in the resulting estimation.

The simplest approaches do not affect the optimality of the full EKF filter. These methods aim to reduce the computation required while still resulting in estimates and covariances that are equal to the full-form SLAM algorithm. The idea is to confine the expensive updates to a small local region and update the global map only at a much lower frequency. Both single map and sub-map approaches exist. The results are optimal SLAM estimate with reduced computation. The computation complexity is usually still $O(N^2)$, but the computation burden is alleviated when scaling to larger environments. Representative work in this category include *compressed EKF (CEKF)* by Guivant and Nebot [63], *postponement* by Knight and Davison [23, 68], *local map sequencing* by Tardós et al. [97], and *constrained local submap filter (CLSF)* by Williams [110].

A second approach examines the possibility of applying sub-optimal updates, so as to provide speedier filtering at the cost of accuracy. Generally the methods work by dividing the map into regions (submaps) and neglecting or approximating some of the coupling between map estimates. These approaches generally provide a conservative estimate in the global frame, i.e., they produce larger uncertainty or covariance than that of the optimal results. Usually the conservative algorithms, while less accurate (due to neglect or approximation of coupling), are computationally more efficient than the optimal method which maintains the optimality of EFK solution. These methods can produce linear $O(N)$ or even constant time $O(1)$ computational complexity. These sub-optimal methods come in two fundamental varieties: globally referenced and locally referenced. Global submap methods estimate the global locations of submap coordinate frames relative to a common base frame. Some of the original work in this field includes *decoupled stochastic mapping (DSM)* [74] by Leonard et al. and *constant time SLAM (CTS)* [75] methods by Leonard et al.. The locally referenced approach is different in that there is no common coordinate frame. The local maps are connected in a graphical network, resulting in a hybrid metric/topological representation. This is the approach taken by the *Atlas* framework by Bosse et al. [15, 16], *Hierarchical SLAM* by Estrada et al. [47], and *Network coupled feature maps (NCFM)* by Bailey [7]. One of the appealing aspects of a hybrid metric/topological approach to mapping and localization is that uncertain state estimates

do not need to be referenced to a single global reference frame.

A third group of methods exploit the fact that reformulation of the standard state-space SLAM representation into information (canonical) form allows sparsification of the resulting information matrix to be exploited in reducing computation. Both optimal and conservative variations exist of these sparsification algorithms. Notable work here includes the *Sparse Extended information filter (SEIF)* [101] by Thrun et al., the *Thin junction tree filter (TJTF)* [84] by Paskin and the *treeMap filter* [55] by Frese. Closely related are batch (offline) algorithms in which a graphical model of the problem is deployed, including graphicalSLAM [51, 52], graphSLAM [102, 99] and relaxation algorithms [31, 56, 57].

It is impossible to review all the work associated with metric/probabilistic approaches to SLAM. Recent reviews can be found in [98, 99, 42, 8]. In summary, 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 computational complexity, data association, and linearity issues.

4.3 Probabilistic approaches to topological mapping

There exists some work that directly extends the above probabilistic approach to topological mapping by computing the probability distribution over the space of all topological maps. This work includes [24], [21], [103], and a series of work by Ranganathan et al. [87, 88, 90, 89]. One of the early works by De et al. [24] uses an E-M algorithm to generate topological maps that are consistent with the data. They incorporate a model selection criterion to penalize over-fitting with notable success. Their approach is only applicable to graphs that have one or two cycles.

[103] presents a method for topological SLAM that specifically targets loop closing for edge-ordered graphs. Similar to the oracle-less work by Dudek et al. [33, 39] (described in the last chapter), the 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 [39]), this paper proposes a probabilistically grounded multi-hypothesis technique. The contribution of the work includes 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 to reduce the size of the hypothesis tree. Loop closing is introduced automatically within the tree expansion, and likely hypotheses are chosen based on their posterior probability after a sequence of sensor measurements. At time step k, each hypothesis h stores the robot's state on that graph, X_k^h, as well as a possible edge-ordered topological graph, G_k^h. The state is represented by the vertex at which the robot is currently located, v_k^h, as well as the edge from which the robot arrived at that vertex, e_k^h, thus $X_k^h = (v_k^h, e_k^h)$. 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 such

as $L_k^h(v_k^h)$ stores the vertices in the graph that are neighbors of vertex v_k^h in the order they occur (counterclockwise 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 $k-1$). The tree expansion algorithm expands all H_{k-1} leaf nodes of the hypothesis in the following way. If $L_{k-1}^h(v_{k-1}^h, \beta_k)$ (the neighbor of v_{k-1}^h 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 $L_{k-1}^h(v_{k-1}^h, \beta_k)$ is *unexplored* then the algorithm considers several possibilities that would 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 proposed approach computes the posterior probability of each hypothesis given a sequence of sensor measurements. The hypothesis that better fits the sensor data would produce 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_k^e is obtained during the edge traversal, which includes a travel distance measurement. Also a measurement z_k^v 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 $p(X_k^h, G_k^h | z_{0:k}, u_{1:k})$ where, as before, X_k^h and G_k^h represent the robot's state and graph respectively. $z_{0:k} = (z_{0:k}^v, z_{1:k}^e)$ is the collection of all measurements during the experiment, which includes the edge measurement sequence $z_{1:k}^e$ and the vertex measurement sequence $z_{0:k}^v$. The sequence $u_{1:k}$ represents the motion inputs through time step k. Using Bayes' law, the posterior is reformulated to $p(X_k^h, G_k^h | z_{0:k}, u_{1:k}) = \eta p(z_{0:k} | X_k^h, G_k^h, u_{1:k}) p(G_k^h | u_{1:k})$ where $p(z_{0:k} | X_k^h, G_k^h, u_{1:k})$ is the measurement likelihood function and $p(G_k^h | u_{1:k})$ is a prior on the hypothesis. The prior $p(G_k^h | u_{1:k})$ is reduced from $p(X_k^h | G_k^h, u_{1:k}) = p(X_k^h | G_k^h, u_{1:k}) p(G_k^h | u_{1:k})$, because the probability of the state given the map and inputs, $p(X_k^h | G_k^h, u_{1:k})$, is equal to one – due to the assumption that a robot can correctly performs the motion input sequence. The scalar value η is used for normalization over possible hypotheses such that $\sum_{h=0}^{H_k - 1} p(X_k^h, G_k^h | z_{0:k}, u_{1:k}) = 1$, where H_k is the number of current leaf nodes in the hy-

pothesis tree. In computing the measurement likelihood function $p(z_{0:k}|X_k^h, G_k^h, u_{1:k})$ 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 covariances for the edge and vertex respectively. Using these notations, the likelihood update is computed recursively using a customized method (see [103] for details). For the prior $p(G_k^h|u_{1:k})$, which represents the probability that the robot happens to be placed in an environment with a topology G_k^h (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 $p(G_k^h|u_{1:k}) \propto \exp(-N_k^h \log k)$. 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 is effectively trying to capture a balance between small concise maps that would make sense for a structured environment and large maps that better fit the data.

In executing the tree expansion algorithm, to keep the tree size bounded, a series of pruning tests are applied to the leaf hypotheses at each time step. In order to reduce the chance of eliminating the hypothesis that represents the true map/state, only conservative pruning rules are applied. A) Degree Test. In the tree expansion algorithm, when $L_{k-1}^h(v_{k-1}^h, \beta_k)$ is *unexplored*, the hypothesis tree adds a child hypothesis for every possible loop closure to any vertex v that also has an unexplored edge. If the detected degree of the arrival vertex is unequal to the degree of vertex v, then that child hypothesis is immediately discarded. This test involves no risk of eliminating the true hypothesis. B) Likelihood Update Test. When updating the likelihood for a new hypothesis recursively, observe whether the updated likelihood exceeds a 4-sigma error bound. If true, this would imply that the new measurements z_k^e and/or z_k^v do not agree with the measurements already associated to the corresponding edge/vertex and are therefore outliers in the data. This further signifies an incorrect loop closure and thus the hypothesis is pruned. This test has an extremely small but nevertheless non-zero chance of eliminating the true hypothesis. C) Planarity Test. Use a strict planarity test ([104]) to eliminate hypotheses that are not planar. This test can often prune a large number of hypotheses without the risk of discarding the correct hypothesis. D) Posterior Probability Test. The last pruning rule is 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 the 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 oracle-less algorithm [33] removes hypotheses in the

tree only when the graph becomes inconsistent, if the implementations in [33] were run on the same data set then the number of hypotheses is expected to grow beyond what is computationally feasible.

Ranganathan et al. proposed probabilistic approaches to topological mapping in a series of papers [87, 88, 90, 89]. [87] introduces the concept of Probabilistic Topological Map (PTM) and provides an algorithm for obtaining a PTM using MCMC sampling. Probabilistic Topological Maps (PTMs) is a sample-based representation that captures the posterior distribution over all possible topological maps given the available sensor measurements. The key realization of this paper is that a distribution over this combinatorially large space can be approximated by a sample set drawn from this distribution. As the second major contribution the paper shows how to perform inference in the space of topologies given uncertain sensor data from the robot, the outcome of which is exactly a PTM. Specifically, the proposed approach uses Markov chain Monte Carlo (MCMC) sampling [59] to extend the Bayesian probabilistic framework to the space of topological maps. The paper defines a PTM to be a data structure that approximates the posterior distribution $P(T|Z)$ over topologies T given measurements Z. In this paper Z contains odometry measurements only (in the authors' later work Z also includes the appearance measurements obtained from the landmark locations). A PDF over the space of possible topological maps is approximated by drawing samples from the distribution over possible maps. The samples are obtained using Markov chain Monte Carlo sampling. To infer the PTM from the measurements, the approach exploits the equivalence between topologies of an environment and set partitions of landmark measurements, which group the measurements into a set of equivalence classes. When all the measurements of the same landmark are clustered together, this naturally defines a partition on the set of measurements. To perform inference, the approach uses the Metropolis-Hastings (MH) algorithm [64] which is a very general MCMC method. All MCMC methods work by generating a sequence of states from a Markov chain, with the property that the generated states are samples from the target distribution. Here the state space that is sampled over is the space of set partitions, where each partition represents a different topology of the environment.

The MH algorithm uses a *proposal distribution* to propose the next state in the Markov chain at each step (a move in state space). Intuitively, the algorithm samples from the desired probability distribution $P(T|Z)$ by rejecting a fraction of the moves generated by the proposal distribution $Q(T_t \rightarrow T'_t)$ where T_t is the current state and T'_t is the proposed state. The fraction of moves rejected is governed by the acceptance ratio $a = \frac{P(T'_t|Z^t)Q(T_t \rightarrow T'_t)}{P(T_t|Z^t)Q(T'_t \rightarrow T_t)}$. Computing the acceptance ration a (hence sampling using MCMC) requires the design of the proposal density Q and evaluation of the target density P. The proposal distribution operates by proposing one of two moves, a *split* or a *merge*, with equal probability at each step. The merge move merges two randomly selected sets in the partition to produce a new partition with one less set than before. The split move splits a randomly selected set in the partition to produce a new partition with one more set than before. Probabilities of the moves are calculated by calculating the proposal

ratio (see [87] for details). Note that this proposal distribution does not incorporate any domain knowledge but uses only the combinatorial properties of set partitions to propose moves. The second task is to evaluate the (target) posterior probability $P(T|Z)$ for each proposed topology change. Using Bayes' law, $P(T|Z) \propto P(Z|T)P(T)$, where $P(T)$ is a prior and $P(Z|T)$ is the observation likelihood. In this work a non-informative uniform prior over all topologies is assumed (but the authors note that it is also possible to use a Poisson distribution on the number of landmarks in the environment if some evidence for this exists.) The authors claim that it is not possible to evaluate the likelihood $P(Z|T)$ without knowledge of the landmark locations. The proposed approach thus integrates over the set of landmark locations X to calculate the marginal distribution $P(Z|T)$ from the joint distribution $P(Z, X|T)$. The likelihood $P(Z|T)$ is then given as $P(Z|T) \propto \int_X P(Z|X,T)P(X|T)$ where $P(Z|X,T)$ is the measurement model, which is defined as an arbitrary density on Z given X and T, and $P(X|T)$ is the prior on landmark locations. Realizing that it is not possible in general to use any form of analytical evaluation to compute $P(Z|T)$, the authors use a Monte Carlo approximation to evaluate the integral, i.e., uses *importance sampling* to approximate the integrand $P(Z|X,T)P(X|T)$. Given a target distribution to be sampled, importance sampling requires a proposal distribution from which samples are actually obtained. Subsequently, these samples are weighted by their "importance", i.e., the ratio of the target distribution to the proposal distribution at the sample point. The weighted samples can then be used in Monte Carlo integration. In the work the importance sampling proposal distribution is an approximation of the log-likelihood. (See [88] for details.) In the experiments the correct topology receives a large probability mass in the distribution obtained with the appropriate parameters.

The above work [87] considered the case where the measurements consist of odometry measurements alone. However, the use of appearance information was not utilized in that work. Further, the proposal distribution used to mix the Markov chain in state space was constructed using a simple split-merge algorithm that does not take into account any domain knowledge, leading to slow mixing and inefficiency in some cases. [88] by the same authors addresses both the shortcomings of [87]. First, it presents a general model for incorporating appearance measurements in the construction of PTMs. As a second extension, the paper proposes a new data-driven proposal distribution for use in the MCMC sampler. The proposal uses domain knowledge in the form of expected landmark locations. This leads to faster mixing of the Markov chain, thus making the PTM algorithm more efficient. Taking appearance measurements into consideration, the set of measurements Z thus consists of odometry measurements O and appearance measurements A, i.e., $Z = \{O, A\}$. The authors note that since the odometry and appearance measurements are conditionally independent (given the topology T), $P(T|O,A) = \eta P(O,A|T)P(T) = \eta P(O|T)P(A|T)P(T)$ is obtained. The evaluation of the odometry likelihood $P(O|T)$ was addressed in [87]. The paper here deals with modeling appearance to evaluate the appearance likelihood $P(A|T)$. The proposed approach uses the Fourier signature of a panoramic image as the appearance measurements in the appearance model. Fourier signatures are a low-dimensional representation of images

using Fourier coefficients. They allow easy matching of images to determine correspondence. The property is exploited when determining correspondence since the robot may be moving in different directions when the images are obtained. (See [88] for details.) The paper also presents a data-driven proposal that significantly speeds up the algorithm by enabling rapid mixing of the Markov chain. The approach is based on the split and merge moves as in [87] but also uses domain knowledge in the form of expected landmark locations. Experiments show that addition of appearance information improves the results significantly by disambiguating noisy odometry. In the experiments the ground truth topology, which received a low probability mass when using only odometry measures (due to noisy odometry), receives high probability mass when appearance measurements are also used. Due to the newly developed data-driven proposal, an improvement in running time is also observed.

Later work [90] combines the work of [87] and [88] by providing a general theory for incorporating both odometry and appearance measurements in the inference process. [89] adds to the previous work by providing a means to compute the PTMs incrementally using particle filtering. While the state space is combinatorial in nature, efficient computation is made possible by Rao-Blackwellization using the location of the landmarks. A data-driven proposal distribution is used for fast convergence.

4.4 Summary

This chapter reviewed metric-based approaches to the SLAM problem. As an alternative to the oracle-based approaches, the probabilistic metric-based approach can be considered as associating pose (metric) information with each node and edge in the topological representation. Traditionally, metric approaches use probabilistic concepts to explicitly represent and manipulate spatial uncertainty. In SLAM the task is modeled as estimating a posterior probability distribution over all possible states, i.e., all possible maps and all possible robot poses, given the controls and sensor readings accumulated by the robot. This distribution is called the *SLAM posterior*. The Kalman filter [99, 66] and the particle filter [99, 53] are important techniques to approximate the SLAM posterior. In general, the probabilistic approaches to SLAM with metric information work well under some reasonable assumptions about environmental complexity.

This chapter also reviewed some recent work that directly extends the probabilistic approach to topological mapping by computing the probability distribution over the space of all topological maps. In contrast to the (deterministic) oracle-based approaches, in the probabilistic approach geometric information such as edge length is maintained.

5 Summary and open problems

5.1 Summary

This report reviews the problem of robotic exploration and mapping in unknown environments. This problem addresses two interrelated problems: mapping and localization. When the robot does not have access to a map of the environment, nor does it know its own pose, the two problems have to be solved concurrently, leading to the SLAM problem. During exploration, a critical task is to disambiguate the current place of the robot against known locations. Within the topological formalism, this is not generally possible, unless some other aids are employed. One approach is to resort to an 'oracle' that can help the robot solve the disambiguation problem. With the help of a sufficiently powerful oracle, the SLAM problem can be solved deterministically. Different oracles have different powers leading to different exploration approaches (see Table 3.6.). One of the challenges of this approach is that employing an oracle is generally expensive, and might not always be feasible. Another approach is to exploit additional information describing the underlying environment. One effective and practical choice is to associate metric information with each robot motion, resulting in metric representations. Within metric formalism, a common approach to solving the mapping problem is to cast the problem within a probabilistic framework, which encodes different sources of uncertainties. The disambiguation task becomes the process of estimating the likelihood of different locations under the associated probabilistic functions. Metric approaches are challenged by several problems, including the scaling problem. Work in these two paradigms are reviewed in Chapter 3 and Chapter 4 respectively.

5.2 Some open problems

The discussions in this report imply a number of open problems. For example, in the oracle-based paradigm, which assumes a graph-like world, one fundamental problem is, given a real world to explore, how to build topological representations for the environment. For complicated, unstructured, or even outdoor environments, this problem is not trivial. As another example, using a super-glue pebble, a directional movable pebble, or footprints oracle have not been fully investigated, especially when exploring directed graphs. Moreover, using other oracles such as a string has not been fully investigated. In the metric SLAM paradigm, open problems include addressing the scaling problem

of SLAM, and SLAM in unstructured environments. There are also some open problems that relate to both paradigms. For example, is it possible to develop oracle-based probabilistic approaches? Suppose a marker is added into the Bayesian framework of the metric approaches, how can this oracle collaborate with the probabilistic framework, such that the problem is more tractable? As another example, the oracle-based approaches reviewed in Chapter 3 are all based on the assumption that the robot can always recognize when it arrives at a vertex, can correctly traverse edges, and can enumerate edges at each node. An interesting open problem is how to build a probabilistic function for potentially erroneous perception (e.g., edge enumeration, node/edge recognition), and incorporate this into the algorithm, so that the algorithm is robust to these errors.

Exploring with other environment properties

Another set of open problems which is more closely related to SLAM in topological representation, is how to explore with other *a priori* knowledge of the underlying environment. Among the work reviewed in this report, there is some work that investigates or discusses the impact of different amounts of *a priori* environment knowledge on the performance of an exploration algorithm. Under the paint can model, [83, 85] investigate how different map information can help exploration, and shows that the more map information available, the easier (less edge traversals) the exploration task would be. Dudek et al. stressed in their oracle-less work [32, 33] the importance of exploiting some environment property information. This information is used as a cue to cease exploration even though some possible models have not been fully explored. An example cue is the *a priori* knowledge of the number of locations 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 at least N vertices. Another cue mentioned in the paper is the *planarity* of the world being explored. Here we list some other possible *a priori* knowledge of the environment properties that might be interesting:

- P(D): The probability that the diameter of the graph is D is P(D).
- P(node degree): The probability of different node degrees. E.g., probability of a node with degree 4 is 30%, and that with degree 2 is 40%.
- P(number of nodes): A probability distribution on the number of nodes in the environment, e.g., number of nodes N is a Gaussian distribution function with mean j and standard deviation of k.

In Chapter 3, we show that in the topological formalism, i.e., exploring an embedded graph, if no oracle is employed, then even if we can assume perfect sensing and motion, no deterministic algorithm exists. Under this situation, an interesting question is how to exploit environment property information so the problem is solvable. In Chapter 4, we show that the metric approaches are challenged by several problems, e.g., the computational complexity and data association problem. Now the interesting question here is

how to incorporate such information into the metric framework such that the problem is more tractable.

Bibliography

[1] Theseus by Plutarch, Written 75 A.C.E. Translated by John Dryden.

[2] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[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, 2004.

[4] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *the 34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 638–647, 1993.

[5] B. Awerbuch, M. Betke, R. L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[6] B. Awerbuch and S. G. Kobourov. Polylogarithmic-overhead piecemeal graph exploration. In *Eleventh annual conference on Computational learning theory (COLT)*, pages 280–286, 1998.

[7] T. Bailey. *Mobile robot localization and mapping in extensive outdoor environments*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, 2002.

[8] T. Bailey and H. Durrant-whyte. Simultaneous Localization and Mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine*, 13(3):108–117, 2006.

[9] 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(2):143–162, 2007.

[10] 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):1–21, 2002.

[11] 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*, pages 75–85, 1994.

[12] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2-3):231–254, 1995.

[13] 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*, pages 132–142, 1978.

[14] 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 (SFCS)*, pages 147–161, 1977.

[15] M. Bosse, P. Newman, J. Leonard, and S. Teller. An Atlas framework for scalable mapping. In *IEEE International Conference on Robotics and Automation*, pages 1899–1906, 2003.

[16] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simulatenous localization and map building in large-scale cyclic environments using the Atlas framework. *International Journal of Robotics Research*, 23(12):1113–1139, December 2004.

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

[18] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *National Conference on Artificial Intelligence*, Portland, OR, USA, 1996.

[19] J. A. Castellanos and J. D. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[20] 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, 2007.

[21] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

[22] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretic Computer Science*, 385(1-3):34–48, 2007.

[23] A. J. Davison. *Simultaneous Localization and Map-Building Using Active Vision*. PhD thesis, Oxford University, 1998.

[24] A. De, J. Lee, and N. J. Cowan. Toward SLAM on graphs. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008.

[25] X. Deng, E. Milios, and A. Mirzaian. Robot map verification of a graph world. *Combinatorial Optimization*, 5(4):383–395, 2001.

[26] 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.

[27] X. Deng and A. Mirzaian. Competitive robot mapping with homogeneous markers. *IEEE Transactions on Robotics and Automation*, 12(4):532–542, 1996.

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

[29] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. *Theoretical Computer Science*, 326(1-3):343–362, 2004.

[30] G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[31] T. Duckett, S. Marsland, J. Shapiro, and S. England. Learning globally consistent maps by relaxation. In *IEEE International Conference on Robotics and Automation*, pages 3841–3846, 2000.

[32] 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.

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

[34] 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.

[35] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Using multiple markers in graph exploration. In *SPIE Vol. 1195 Mobile Robots IV*, 1989.

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

[37] 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.

[38] 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.

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

[40] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. *ACM Transactions of Algorithms*, 2(3):380–402, 2006.

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

[42] H. Durrant-whyte and T. Bailey. Simultaneous Localization and Mapping (SLAM): Part I. *IEEE Robotics and Automation Magazine*, 13(2):99–100, 2006.

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

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

[45] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[46] A. Eliazar and R. Parr. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1135–1142, Acapulco, Mexico, 2003.

[47] C. Estrada, J. Neira, and J. D. Tardós. Hierarchical SLAM: real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, 2005.

[48] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *13th Annual International Symposium on Algorithms*, pages 11–22, 2005.

[49] 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.

[50] P. Flocchini, B. Mans, and N. Santoro. Sense of direction: Definitions, properties and classes. *Networks*, 32(3):165–180, 1998.

[51] J. Folkesson and H. Christensen. Graphical SLAM - a self-correcting map. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 383–390, 2004.

[52] J. B. Folkesson and H. I. Christensen. Robust SLAM. In *International Symposium on Autonomous Vehicles (IAV)*, 2004.

[53] 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.

[54] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

[55] U. Frese. Treemap: An *O(log n)* algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122, 2006.

[56] U. Frese and T. Duckett. A multigrid approach for accelerating relaxation-based SLAM. In *IJCAI workshop on Reasoning with Uncertainty in Robotics (RUR2003)*, pages 39–46, Acapulco, Mexico, 2003.

[57] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, 2005.

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

[59] W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, 1995.

[60] J. Grimm, W. Grimm, and J. Zipes. *The Complete Fairy Tales of the Brothers Grimm All-New Third Edition*. Bantam Books, New York, N.Y, 2003.

[61] G. Grisetti. Towards a PhD Thesis on simultaneous localization and mapping, 2005.

[62] J. L. Gross and T. W. Tucker. *Topological graph theory*. Wiley-Interscience, New York, NY, USA, 1987.

[63] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotic and Automation*, 17(3):242–257, 2001.

[64] W. Hastings. Monte Carlo samping methods using Markov chains and their applications. *Biometrika*, pages 97–109, 1970.

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

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

[67] J. Kemeny and J. Snell. *Finite Markov Chains*. D. Van Nostrand Co., Inc., USA, 1960.

[68] J. Knight, A. Davison, and I. Reid. Towards constant time SLAM using postponement. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 406–412, 2001.

[69] 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.

[70] B. Kuipers, D. Tecuci, and B. Stankiewicz. The skeleton in the cognitive map: A computational and empirical exploration. *Journal of Environment and Behavior*, 35(1):81–106, 2003.

[71] 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.

[72] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[73] J. J. Leonard, H. F. Durrant-Whyte, and I. J. Cox. Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Research*, 11(4):286–298, 1992.

[74] J. J. Leonard, H. Jacob, and S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *9th International Symposium on Robotics Research*, 1999.

[75] J. J. Leonard and P. M. Newman. Consistent, convergent, and constant-time SLAM. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1143–1150, Acapulco, Mexico, 2003.

[76] J. Liu, R. Chen, and T. Logvinenko. A theoretical framework for sequential importance sampling and resampling. In A. Doucet, N. De Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer Verlag, New York, 2001.

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

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

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

[80] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. In *In 5th International. Symposium on Robotics Research*, pages 207–216, Tokyo, Japan, 1989.

[81] P. Newman. *On the Structure and Solution of the Simultaneous Localization and Map Building Problem*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, 2000.

[82] P. Panaite and A. Pelc. Exploring unknown undirected graphs. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 316–322, 1998.

[83] P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36(2):96–103, 2000.

[84] M. A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1157–1164, San Francisco, CA, 2003.

[85] L. M. Paz and J. Neira. Optimal local map size for EKF-based SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5019–5025, Beijing, China, 2006.

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

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

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

[89] 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, 2006.

[90] A. Ranganathan, E. Menegatti, and F. Dellaert. Bayesian inference in the space of topological maps. *IEEE Transactions on Robotics*, 22(1):92–107, 2006.

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

[92] W. J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences (JCSS)*, 7(4):389–403, 1973.

[93] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

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

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

[96] H. Sorenson. Least-squares estimation: from Gauss to Kalman. *IEEE Spectrum*, 7:63–68, 1970.

[97] J. D. Tardós, J. Neira, P. M. Newman, and J. J. Leonard. Robust mapping and localization in indoor environments using sonar data. *International Journal of Robotics Research*, 21(4):311–330, 2002.

[98] 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.

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

[100] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

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

[102] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[103] 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)*, to appear, 2009.

[104] G. Vijayan and A. Wigderson. Plananrity of edge ordered graphs. Technical Report TR307, Princeton University, 1982.

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

[106] 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, 2008.

[107] 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)*, Phuket, Thailand, 2009.

[108] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill, 1995.

[109] 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.

[110] S. Williams. *Efficient solutions to autonomous mapping and navigation problems*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, 2001.