5-1-2013

Hopscotch: Robust Multi-agent Search

Edward C. Miles

Edward C. Miles

Candidate

Computer Science

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Melanie Moses , Chairperson

Lydia Tapia

Rafael Fierro

Hopscotch:
Robust Multi-agent Search

by

Edward C. Miles

B.E.E., Georgia Institute of Technology, 1990

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May 2013

Acknowledgments

Thanks to:

My professors and fellow students at UNM, who contributed to the vat of ideas that eventually distilled into this thesis.

The Scalenet lab group, who provided a welcoming forum for the exchange of ideas and inspiration.

My committee, and especially my advisor, Melanie Moses. She kept me focused and motivated and provided notebooks full of advice, most of which I followed.

Hopscotch:
Robust Multi-agent Search

by

Edward C. Miles

B.E.E., Georgia Institute of Technology, 1990

M.S., Computer Science, University of New Mexico, 2013

Abstract

The task of searching a space is critical to a wide range of diverse applications such as land mine clearing and planetary exploration. Because applications frequently require searching remote or hazardous locations, and because the task is easily divisible, it is natural to consider the use of multi-robot teams to accomplish the search task. An important topic of research in this area is the division of the task among robot agents. Interrelated with subtask assignment is failure handling, in the sense that, when an agent fails, its part of the task must then be performed by other agents.

This thesis describes Hopscotch, a multi-agent search strategy that divides the search area into a grid of lots. Each agent is assigned responsibility to search one lot at a time, and upon completing the search of that lot the agent is assigned a new lot. Assignment occurs in real time using a simple contract net. Because lots that have been previously searched are skipped, the order of search from the point of view of

a particular agent is reminiscent of the progression of steps in the playground game of Hopscotch.

Decomposition of the search area is a common approach to multi-agent search, and auction-based contract net strategies have appeared in recent literature as a method of task allocation in multi-agent systems. The Hopscotch strategy combines the two, with a strong focus on robust tolerance of agent failures. Contract nets typically divide all known tasks among available resources. In contrast, Hopscotch limits each agent to one assigned lot at a time, so that failure of an agent compels re-allocation of only one lot search task. Furthermore, the contract net is implemented in an unconventional manner that empowers each agent with responsibility for contract management. This novel combination of real-time assignment and decentralized management allows Hopscotch to resiliently cope with agent failures.

The Hopscotch strategy was modeled and compared to other multi-agent strategies that tackle the search task in a variety of ways. Simulation results show that Hopscotch is failure-tolerant and very effective in comparison to the other approaches in terms of both search time and search efficiency. Although the search task modeled here is a basic one, results from simulations show the promise of using this strategy for more complicated scenarios, and with actual robot agents.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 General Considerations

The challenge of searching a space thoroughly and efficiently is applicable to a wide variety of tasks such as land mine detection [1], resource mapping [2], and planetary surface exploration [3]. Rather than using a small number of very reliable robot agents to explore a space, as with the rovers recently deployed to Mars, there are numerous advantages to using a large number (a "swarm") of simple robots [4]. This is analogous to biological swarms such as those of ants or bees, where numerous foragers search an area for resources [5].

The goal of the search process depends on the application. The goal may be to find a resource that can be exploited, to determine a safe path, or to map the positions of obstacles. In cases like these, the search can be successful without being exhaustive, since the search can end when the goal is attained. For applications such as land mine clearing, it may be desirable or even required that the search completely cover

the entire search space [6]. This goal of complete coverage is common for analogous problems appearing in the literature such as *cleaning*, *painting*, and *mowing* [7, 8].

Many multi-agent strategies have been proposed for this type of task [9], often without consideration of agent reliability. Strategies that rely on each agent performing a specific subtask assigned a priori (i.e., searching a particular area) are especially vulnerable to agent failures [10]. Strategies that incorporate redundancy in agent tasking are less vulnerable to failures but suffer the inefficiencies inherent with redundancy. Ideally, a strategy should work quickly, be fault tolerant, and completely cover the search area.

In order to quickly search large areas, it is generally desirable to have as many agents participating in the search as possible. However, economics and logistics place limits that result in a trade-off between systems comprised of a few highly capable search agents and systems with more agents that are less capable (where less capable implies smaller, cheaper, more transportable, and perhaps less reliable) [11]. In the latter case, the agents may cooperate, forming a *swarm*. There are numerous examples from biology of species successfully using swarms of simple agents to accomplish complex tasks [12]. The foraging strategies of harvester ant colonies have been studied extensively, and even these simple agents, taken collectively, exhibit complex and efficient strategies for resource exploitation [13].

With the goal of realizing these types of benefits, there has been quite a bit of recent work devoted to developing man-made swarm search systems [14, 4]. These systems are usually comprised of a number of small, simple robot agents. A significant advantage of this approach is that, should an agent fail, remaining operational agents can compensate for the loss. One of the main challenges in designing a swarm search strategey is subtask allocation [15].

Because use of robots for search is well suited for remote or hazardous locations,

there will be many search tasks where the robot swarm must operate autonomously. In this situation, the only influences on the behavior of an individual agent are from the environment, meaning the space and other agents. Furthermore, it means that an agent that fails cannot be repaired. While there has been work suggesting the feasibility of self-repair within a robot swarm [16, 17], this type of functionality is likely to require capabilities beyond what is considered here for simple agents.

1.1.2 Problem Description

This thesis describes simulation and testing of different strategies for searching a space using multiple unreliable agents. Although the experimental setup is not intended to precisely represent a particular real-world application, efforts were made to keep the experiments realistic enough that the results could be applied to real applications. Examples of tasks for multiple agents searching a large, obstacle-free space include land mine detection, surveillance and mapping using unmanned aerial vehicles, and sowing seed on a large field.

This work specifically considers the problem of searching a known, homogenous, obstacle-free area using a swarm of agents that fail at a fixed rate. While the bounds of the search area are known a priori, not all strategies considered herein make use of that knowledge. Furthermore, it is assumed that the agents have accurate location information, though it not all strategies require it. When required by the strategy, it is assumed that the agents can communicate with each other (refer to section 2.2.2 for a detailed description of each strategy). The areas used for the simulation experiments described herein are square, and agents start each simulation run at a centrally located *nest*. The goal is to completely search (or, equivalently, clean) the entire space, or some predetermined fraction of the space.

More formally, the search task is defined as follows. The search *space* is a

bounded, fully connected planar region. The space is divided into a grid of equal-sized axis-aligned square *locations*. One location is designated the nest, and serves as the source for search agents entering the space.

Time is discretized into steps. At each time step, each agent is given the opportunity to either change direction or move discretely in the current direction to a new location. Agents may not move to a location that is already occupied. Each agent has sensors that allow it to search the location that it occupies, and the size of a location is considered to be the area an agent can search in a single time step. A location is considered to have been searched during the time step that an agent moves to that location. Each agent is also assumed to have the ability to detect obstacles such as boundaries and other agents that occupy neighboring locations. Furthermore, each agent has a chance of failing each time step. Failed agents do not participate in the task in any way and do not recover.

Each agent operates based on a set of pre-programmed rules, and is assumed to have a small, finite memory, capable of holding internal state information. This may be used to retain information such as direction, destination, or the last several locations occupied. Furthermore, agents may communicate with one another. The information exchanged between agents depends on the requirements of the search strategy.

Because it is necessary to track what has been searched, both for analysis and as data that affects agent decisions, agent movement is restricted to the four cardinal directions. Since an agent's sensor is considered to operate on an area that is essentially square and the size of a location, allowing movement in a diagonal direction would result in incomplete or imperfect coverage of individual locations. Agents are considered to have accurate knowledge of their current location and orientation (heading), although not all strategies considered use this information. The assumption that robots have accurate location information is a reasonable one and a number

of methods [18, 19] for location determination have been successfully implemented.

1.1.3 Approach

This thesis describes Hopscotch, a self-organized, incremental divide-and-conquer strategy using a swarm of homogenous search agents. Each agent is responsible for a small fraction of the search task at any given time, with assignments handled by a contract net. The contract net is designed so that only one agent is assigned to a given area (a *lot*) in an attempt to minimize redundant searching, which leads to faster coverage of the search space. If an agent fails to meet its obligation to search a lot, the lot becomes available to other agents. Lots that have not been searched are assigned to available agents until all lots have been searched or no agents are available, ensuring that as long as at least one agent remains available, the search will complete. Robustness comes from both managing the scope of work that does not complete should an agent fail, and ensuring that the incomplete work is reassigned.

The Hopscotch strategy is modeled and compared to strategies that rely on redundancy to compensate for agent failures, a strategy that uses individual memory, and strategies that rely on one-way communication using markings such as pheromones. All of the strategies are described in detail in section 2.2.2.

1.1.4 Related Work

Strategies and Multi-agent Considerations

A common application of the search problem is land mine and unexploded ordnance detection in a field, and there is a wealth of published research on this topic. The problem is described by Acar et al. in [20]. While Acar et al. do not consider multiple agents, they do argue the advantages of complete strategies and demonstrate their

advantage over random strategies. Their work to determine an exact cellular decomposition of an unknown search space has applicability to task division in multi-agent strategies. They also address the imperfect detector scenario, where sensing an area that has a search target does not mean the search target is found. Imperfect detectors are especially vexing when complete coverage is desired, since it can never be guaranteed that all search targets (the land mines, in this case) have been found.

There are a number of examples in the literature that illustrate the value of communication when multiple agents are performing a task. Balch and Arkin in [21] explored the impact of communication ability on the capabilities and effectiveness of a multi-agent robotic system. Their research demonstrates that even simple inter-agent communication capability can greatly benefit speed and efficiency. Their experiments included both simulation and real robots, and one of the tasks they studied they refer to as *grazing*, which is a version of the search problem described herein. They consider three levels of communication: no communication, state communication only, and goal communication. Their results show that explicit communication has a negligible effect on the speed of accomplishing the graze task, but they do assume implicit communication, where agents can tell which areas have already been grazed.

In [22], Gage discusses the benefits of randomized search, as compared to coordinated search, for a multiple agent system tasked with coverage. While he does not dispute the benefits of coordinated search strategies, he argues that they require command and control infrastructure, thus increasing the complexity (along with the size, expense, energy requirements, and potential for individual failure) of the system. Also, real-world constraints such as navigation inaccuracies can cause problems for a coordinated strategy.

Gage points out that the application is an important factor in determining how effective a strategy is. For example, the goal of maximizing the number of search targets found per amount of search effort may require a different approach than the

goal of minimizing the number of search targets missed per sweep of an area. He concludes that a random search strategy is not always inferior, and the best search strategy depends on the application parameters.

Biologically Inspired Communication

Koenig et al. in [23] and [24] studied a coordinated multi-robot search strategy that uses very simple ant robots whose only means of communication is leaving markings on terrain that has been searched. The very simple ant robots employing this strategy do not need memory, knowledge of the terrain, or path planning capability. They do not even need to know their location, and they are capable of dealing with situations where the robots are moved without knowing it, e.g., from a collision. The strategy robustly handles robot failures and destruction of communication markings.

Marking as a means of communication between agents has received extensive attention. Wagner and Bruckstein had previously proposed a similar approach in [25], specifying the application to actually be cleaning a dirty floor, and using the amount of dirt on the floor as the means of inter-robot communication. Oshervich et al. in [26] describe a strategy they refer to as "Mark-Ant-Walk", which uses pheromone markings as the means of communication when coordinating coverage of search spaces.

In [27], Ranjbar-Sahraei et al. describe an approach to multi-robot coverage using pheromone-based communication. Here, coverage means a distribution of robots over an area. In their simulations, robots move to areas based on pheromone encounters, resulting in a fairly even spread of agents once a steady-state has been reached.

Using a novel and biologically inspired approach, Hsieh et al. [28] developed a decentralized algorithm for multi-robot distribution among sites inspired by ant colony nest site selection. Their approach uses a quorum sensing mechanism that does not require explicit wireless communication between agents. The agents are

assumed to have the ability to determine whether a site is above or below quorum based on encounter rates with other agents. They show mathematically and through simulation that their algorithm converges to a desired distribution.

Pre-planned Paths

In cases where the travel time to an area to be searched is significant compared to the time to search the area, it becomes important to develop efficient search paths. Hazon and Kaminka in [29] examine this by defining a graph $G(V, E)$ where V is the set of nodes corresponding to the cells (analogous to lots) to be searched, and E is the paths connecting the cells, weighted by travel time. They then find the minimum spanning tree for G, and use this to analyze both backtracking and non-backtracking search strategies that are both complete (guaranteed to search all lots) and robust (completes the search as long as at least one robot is active). This approach takes advantage of a priori knowledge of the search space in order to build the graph. They show that their backtracking strategy has a better time guarantee, but is not necessarily optimal in all applications.

Roadmap-based path planning is extended by Bayazit et al. in [30] for use by groups to improve performance in group behaviors such as exploring. Flocking behavior is typically modeled using individuals with simple rules that determine how members move in relation one other. By integrating roadmaps, rules may be modified for more optimal behavior based on location on the roadmap. The roadmaps themselves may also be adapted by changing roadmap-embedded rules as new information is gathered. This does require that all agents have the ability to read from and modify the roadmap. Simulation results are presented that show good coverage performance in an environment occupied by obstacles.

Explicit Multi-agent Coordination

Burgard et. al. propose a scheme in [31] that sends individual agents to different "target points" so that agents are simultaneously exploring different regions in the environment, and weighting the value of the the target points based on the amount of proximate unexplored area. The algorithm they propose explicitly coordinates the agents. Their experiments focus on complicated environments where it is necessary for the robots to create a map of the area. In their experiments, uncoordinated agents using a greedy strategy frequently duplicated each other's work, and were significantly outperformed by agents using the proposed coordinated strategy.

Alur et al. [32] developed and tested a set of tools to coordinate and control robots deployed in an unknown environment. Their framework divides overall control into a hierarchy of high-level (long term, planned) and low-level (immediate) tasks, and provides a method of transitioning from task to task. Because of the uncertainty inherent in unknown and unstructured environments, robots operating in such environments need a variety of behaviors to deal with different conditions. They also need a control framework to choose behaviors and switch between them in order to accomplish the top-level objective. They implement their framework in a small group of robotic agents and experimentally show good performance in basic tasks such as mapping and leader following. They also present a method of localization using landmark matching from images obtained using an on-board camera.

Bezzo and Fierro in [33] give algorithms for distributing a robot swarm so that wireless network routers on the robots maintain connections between explorers and a base station as the explorers move about the environment. They present simulation results that demonstrate how connections are maintained when members fail.

Distribution of a group of mobile sensors for optimal sensor coverage is addressed by Cortés in [34]. The goal of this distribution is to achieve sensor coverage, where

in this case coverage means the the sensors are distributed such that overall sensing ability is optimized. Each individual periodically communicates with other agents and computes its own Voronoi region, adjusting its location to the center of its Voronoi region for best coverage. This technique requires reliable communication between agents and accurate knowledge of agent locations.

Bullo et al. [35] expand on the sensor coverage ideas in [34], relaxing communication requirements to allow operation with only *gossip communication*, which they describe as asynchronous, pairwise, and possibly unreliable. This leads to *dominance regions*, which are location-independent and recomputed every time an agent communicates with another agent. They show theoretically and with simulation that dominance regions from their gossip coverage algorithm converge asymptotically to a centroidal Voronoi partition. They include a discussion of how the gossip coverage algorithm could be implemented with a robotic sensor network.

A multi-robot task allocation approach requiring extensive communication capability and central control is given by Howard et al. in [36]. They describe a relatively large (approximately 80 robots) heterogenous swarm designed for building interior exploration. They use a mix of two classes of robots: a small number of highly capable robots, complimented by a large number of simple robots. The robots communicate among themselves and with a remote operator console over 802.11b WiFi, which all robots are equipped to use. Control and data aggregation are handled at the remote operator console.

Using a similar centralized control approach, Berman et al. [37] present a method of allocating tasks in robot swarms that does not require communication among the robots themselves. Their method does, however, require a central controller that is capable of monitoring tasks and broadcasting task information, but does not dictate specific actions. They claim that market-based techniques scale poorly to large-scale systems, with one of the reasons being bandwidth limitations.

Market-based Task Assignment

Smith [38] describes the contract net protocol, a high-level protocol for resource allocation in a distributed problem solver. The contract net is made up of a collection of nodes, with a node serving in the role of *manager* or *contractor*. The manager identifies, monitors, and processes the results of tasks executed by the contractor. The contractor executes tasks based on established contracts. A node may be simultaneously performing the role of manager and contractor for different contracts.

In a contract net, nodes capable of performing a task communicate *bids* to the contracting manager. Managers evaluate the bids, and a *contract* for a task is established by communication between the manager and one (or more) of the bidding nodes, thus allocating the task. The paper describes in detail a protocol for exchanging information between managers and contractors. It also provides insights on the process of evaluating bids and choosing contractors.

Choi et al. in [39] use decentralized auction schemes for allocating tasks to robot agents. An agent submits bids for tasks it can accomplish to other agents it is in communication with. The agents then use a *consensus* strategy where each agent computes a winning bids list and compares it to the lists of other agents, iterating until a winning bids list is agreed upon, resulting in agreed-to assignments for each agent. They extend this strategy to also handle *task bundles*, where each agent bids on a set of tasks.

Sariel and Balch use auctions to assign exploration targets to robot agents in [40]. They consider complicated environments, using auctions to redistribute targets as awareness of the environment improves. They use *precaution routines* to handle failures and ensure that the exploration task completes.

Agent Limitations

The case for using multiple less-reliable robot agents is presented by Stancliff et al. in [41]. They define a simple mission, and show that a team of four robots is more likely to successfully complete the mission than a team of two robots even when the robots in the four robot team are much less reliable. In [10] they extend this work by showing that failing to consider robot failures when planning missions can result in sub-optimal plans with substantially worse performance compared to plans that assume failures will occur.

Rubenstein et al. in [42] present a design for very inexpensive robots they call *Kilobots* intended to be operated in swarms numbering on the order of a thousand. These small robots can turn and move forward, but do not have any type of odometry for self-location. The robots can communicate using infrared light, although the range and bandwidth are limited compared to communication systems on more capable robots described in the literature. They also feature an 8 Mhz processor and 32K of memory. The authors demonstrate some basic collective behaviors, but more importantly, these small robots hint at the types of limitations that are likely to be inherent in robots designed to be part of a large swarm.

Extension of Previous Work

This work considers a problem established in some of the earlier work reviewed above, that of *cleaning* or *covering* an area. With a nod to biological inspiration and applications such as land mine detection, the same basic problem is herein referred to as *search*. The literature makes it clear that multi-agent strategies are well suited to easily divisible problems such as this, and acknowledges the challenge of task division and assignment. Cellular decomposition is a common approach to dividing the space among agents. For task assignment, market-based strategies akin to the

contract net presented by Smith in 1980 have become popular in recent literature. The Hopscotch strategy presented in this thesis uses cellular decomposition to divide the search space, and a simple contract net to assign search responsibility for pieces of the space to agents. Hopscotch also modifies these ideas in two ways, with the intent of robust tolerance to agent failures. First, in contrast to traditional contract nets, each agent only contracts for one task at a time. Second, all agents are empowered with contract management responsibility. Taken together, these two modifications minimize the impact of agent failure.

1.2 Outline

The remaining chapters are organized as follows. Chapter 2 describes the simulation used to generate the data described herein and metrics for evaluating the results. Chapter 3 gives the results and analysis from a number of different runs of the simulation. Chapter 4 gives conclusions and presents ideas for further research.

Chapter 2

Methods

2.1 Model Description

The model is implemented in a program called SwarmExp, which was written by the author specifically to perform the experiments described in this thesis. SwarmExp is written in Objective-C, which is an object-oriented superset of the C programming language [43]. In SwarmExp, each agent is an instance of an agent object. This architecture allows each agent to operate exactly the same way, yet maintain its own independent status information. Agent behavior is governed by a set of rules (the *ruleset*) implementing a strategy that is chosen prior to starting the simulation. Information about the space is updated during the course of the simulation run. The space is divided into a grid of squares representing *locations* in the search space, and the size of each square may be thought of as approximating the sensing area of an agent. The amount of information about the space that is available to each agent depends on the ruleset.

A run begins with each agent and the space being initialized with the parameters for that run. Agents enter the space at a location designated as the *nest*, and move

discretely so that an agent occupies only one location at a time. The simulation proceeds in discrete time steps. What agents do during a time step is dictated by the ruleset associated with the strategy being simulated. Typically they will move to one of the four neighboring grid squares (the von Neumann neighborhood with $r = 1$) [44] or change direction. Agents may not both turn and move in the same time step, and an agent may not move to a location that is already occupied. If the agent does move, the newly occupied location is considered to have been searched. Agents update in a fixed order each time step, and the first agent to update its location to a new location l takes precedence over other agents that attempt to move to l.

A depiction of the space during a typical run is shown in figure 2.1.

2.2 Search Strategies

2.2.1 Rulesets

A ruleset implements a search strategy and is a collection of rules that determines the actions each agent takes during a time step. The ruleset is also specifies how much information from the space and from other agents the agent uses, determined by the inputs to the rules.

Specifically, the ruleset defines:

- When an agent changes direction
- How a new direction is determined
- How an agent reacts if the agent finds its destination occupied.

A flow for an agent update showing where rules are implemented is given in figure 2.2.

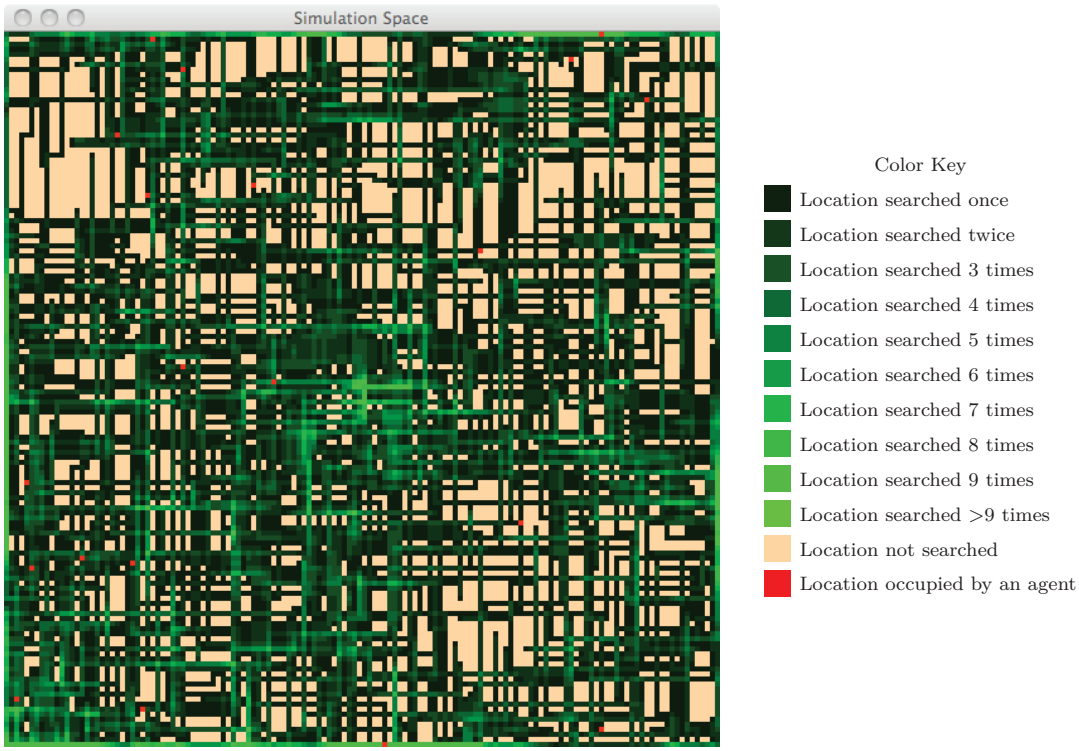


Figure 2.1: Depiction of the simulation space during a typical run. This is the 2000th time step of a run with using 20 agents, with the nest located at the center. Agents locations are marked in red. Unexplored locations are tan. Explored locations are green, ranging from a very dark green if the location has been searched once to a very bright green if the location has been searched 10 or more times.

2.2.2 Description

The search strategies explored in this work are compared in table 2.1. In the table, Comm refers to the type of communication allowed between agents. Space Awareness refers to whether the agent needs to know the extents of the search space, and Location Awareness refers to the agent accurately knowing its own location in the search space. Complete refers to whether the search will necessarily complete given agents remain active.

For the first two strategies, Random Direction Change and Random Chords, each

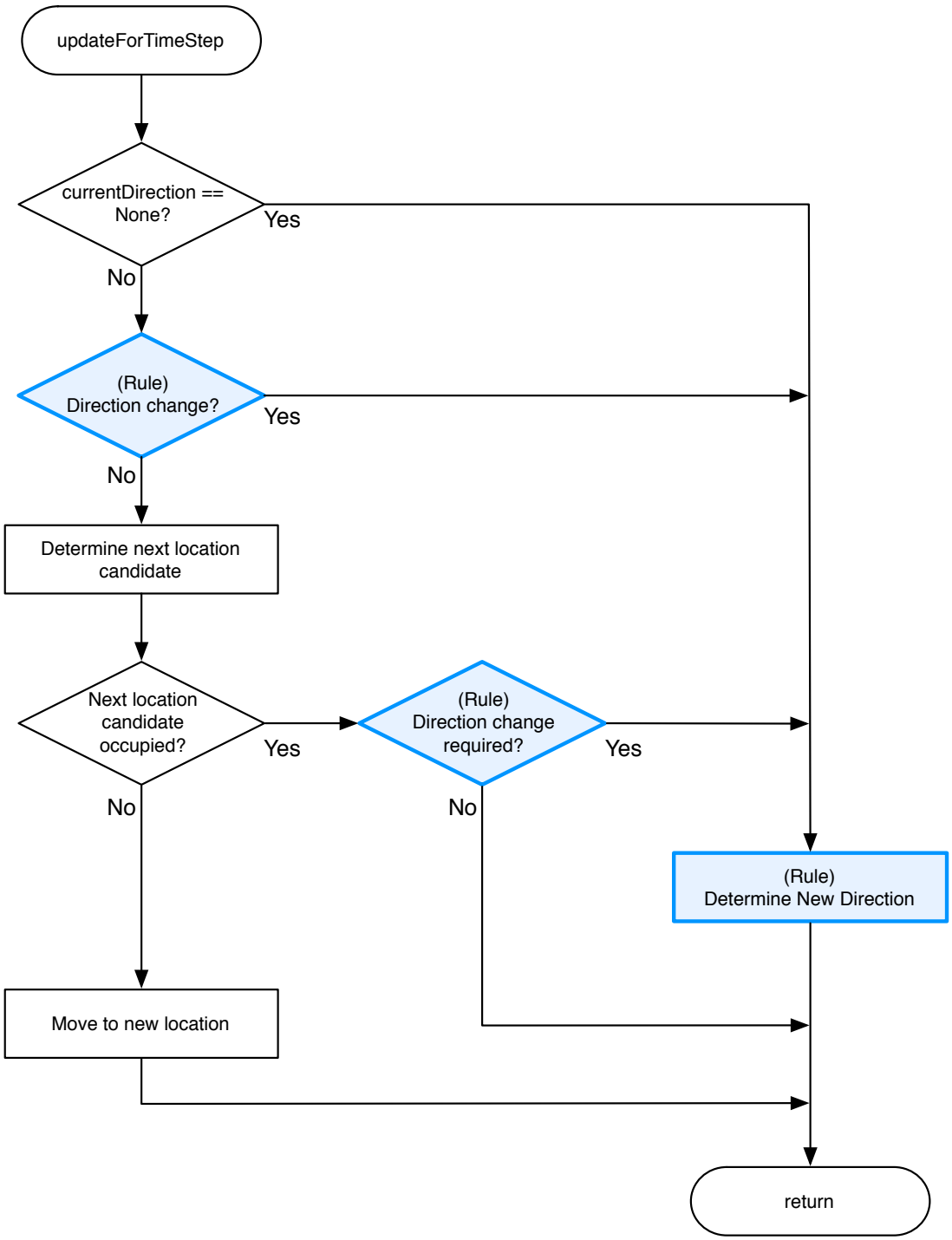


Figure 2.2: Flow of an agent update during a time step. Blue shaded boxes show where rules are implemented.

Strategy	Comm	Space Awareness	Location Awareness	Complete
Random Direction Change	None	Not required	Not required	No
Random Chords	None	Required	Not required	No
Greedy	Markers	Not required	Not required	No
Greedy Lots	Markers	Required	Required	No
Lots With Individual Tracking	None	Required	Required	Yes
Hopscotch	Contract Info	Required	Required	Yes

Table 2.1: Characteristics of tested search strategies.

agent performs its own independent randomized search. For this reason, it is not necessary for agents to communicate with each other or maintain accurate location awareness. The two greedy strategies, Greedy and Greedy Lots, rely on communication by marking locations that are searched. This marking could be implemented with visual marks on the space, pheromone, tags, or in some other manner [45]. The Greedy Lots, Lots With Individual Tracking and Hopscotch strategies divide the space into lots, and agents therefore need to be aware of the extents of the space and their own location to determine where lot boundaries are. Hopscotch requires two-way real time communication between agents.

It is worth pointing out that the table lists just the requirements to execute the search strategy. In a real-world application, more rigorous requirements may be necessary. For example, if the application is a search to determine the location of some set of objectives, agents will need accurate location awareness in order to report where objectives were found. If the goal of the search is surveillance, it may be important for agents to have a communication capability for quickly sending notification if an intruder is found.

Each of these six strategies is described in detail below.

Random Direction Change

The random direction change strategy specifies that each agent has a fixed chance of changing direction at every time step. Upon initialization, all agents have a direction of `None`, so a new direction is determined at random during the first time step. At each subsequent time step, the agent determines whether it should change direction based on a random draw given the fixed chance of changing direction. The agent then either attempts to move in its current direction or changes direction based on the outcome of the draw.

An agent may be unable to move in its current direction due to an obstruction such as a boundary or another agent. If this occurs, the agent changes its current direction to a random direction.

This strategy does not require communications between agents, knowledge of the space to be searched, historical tracking, or accurate location awareness.

Random Chords

Inspired by an algorithm described in [46] that uses chord paths to randomly search a circular area, random chords uses right-angle paths from one side to an adjacent side. Agents initially go to a location at one of the boundaries of the search space. A random location on one of the adjacent boundaries is then picked and the agent travels two right angle segments to reach the new location. Upon arrival, the process of picking a boundary destination and then traveling to it repeats. This is only interrupted if the agent is obstructed by another agent. If this occurs, the agent simply picks a new location on any boundary and travels there, resuming the process.

The Random Chords strategy does not require communication between agents, but does require a knowledge of the space to be searched in order to determine truly

random boundary destinations. However, because it results in long non-crossing paths and involves comparatively little turning, it typically outperforms the Random Direction Change strategy in search time.

Greedy

The Greedy search strategy attempts to maximize time spent searching locations that have not been previously searched, locally optimized both spatially and temporally. The rule assumes that an agent knows whether locations in its local neighborhood have been searched. Though an agent only needs local knowledge for a particular decision, agents could be operating anywhere in the space, meaning that search status for the entire space must be maintained.

If the next location in an agent's current direction has not been searched, the agent moves in its current direction in order to reach that location. If the next location has already been searched, the agent checks its local neighborhood for locations that have not been searched. If a location that has not been searched is found, the agent turns so that its new current direction is toward the location that has not been searched. If all locations in the local neighborhood have been searched, the agent moves in a manner similar to the Random Direction Change strategy until it finds a location that has not been searched, at which time it resumes the greedy behavior.

Implementation of the Greedy strategy requires tracking of which locations in the space have been searched, and having a subset of that information pertaining to its local neighborhood available to each agent. This means that the swarm must have a means of communicating this information. The intent is to simulate this information being communicated using a marker such as pheromone applied to the space at locations that have been searched.

Greedy Lots

The Greedy Lots search strategy uses a decomposition of the space into a grid of square *lots*, with each agent using the same lot boundaries. Like the Greedy strategy, Greedy Lots attempts local optimization but instead of optimizing to locations, entire lots are considered. As with the Greedy strategy, agents are aware of which locations have been searched, and additionally, how many times. This is akin to sensing an accumulation of pheromone or other type of marker. To spread the agents throughout the space initially when there are no previous searches to guide lot choice, a "shotgun start" is used where each agent completely searches a lot chosen at random from all available lots. Once the lot has been searched, the agent evaluates the search status of the eight neighboring lots. Search status is determined by adding the times each location in each respective lot has beens searched, with a small random factor added to simulate sensor noise. The agent uses this value to choose the least searched neighboring lot, searches it, and then chooses subsequent new lots to search in a similar manner.

This strategy requires enough knowledge of the space to subdivide it into lots and a way of communicating search status similar to the Greedy strategy. Though it does require accurate location to determine lot boundaries, a real-world implementation of Greedy Lots should be somewhat tolerant of location errors provided the search markings are applied directly to the floor or ground.

Lots With Individual Tracking

With the space divided into lots, each agent can easily track its own progress and avoid searching where it has already searched. The use of lots provides a decrease in individual agent memory requirements proportional to the lot area. For example, status of a 10×10 lot can be tracked by storing status information with the location

of one corner (given a convention for lot size and which corner is stored), as opposed to storing status information for all 100 locations the lot contains.

Lots With Individual Tracking specifies that each agent chooses a lot and searches it; upon completion of searching the lot, the agent stores in its individual memory that the lot has been searched. In order to spread out the agents, a "shotgun start" is used, sending each agent initially to search a randomly selected lot. The agent then chooses a new lot that it has not searched and proceeds to search the new lot, continuing in this manner until all lots have been searched. The choice of lots after the first has a randomness factor but is heavily biased toward lots that are close to the agent. Because of this bias and the tracking of previously searched lots, the lot search progression resembles a self-avoiding random walk.

This rule does not require communication between agents, but it does require enough knowledge of the space to be searched to divide it into lots, and memory proportional to the size of the space based on lot size as previously described.

Hopscotch

As with the game of Hopscotch found on many playgrounds, the Hopscotch search strategy involves division of the space into lots, which, from the point of view of an individual agent, are either searched or skipped based on their search status. This strategy uses the same lot concept as Greedy Lots and Lots With Individual Tracking, but divides lot search responsibility using a simple contract net. This type of market-based approach that has proven effective in dividing work among robot agents [47].

The bidding process for lot assignment is illustrated in figure 2.3. An agent acquires responsibility (a *contract*) to search a lot by using communications with other agents to broadcast a *bid*. The bid includes a lot, which is chosen from a

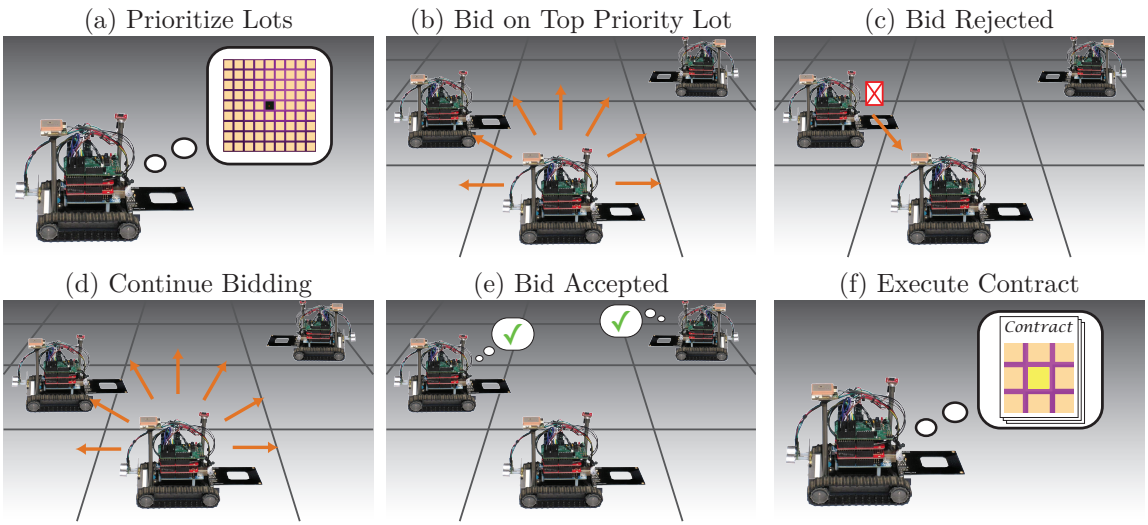


Figure 2.3: The Hopscotch bidding process. (a) The bidding agent assembles a prioritized list of lots to search. (b) The bidding agent broadcasts a bid for its highest priority lot. (c) If another agent is aware that the lot has been searched or is under contract, it rejects the bid. (d) The bidding agent continues to bid on lots in priority order. (e) When the bidding agent does not receive a rejection response, the bid is considered accepted. (f) The bidding agent is now contracted to search the lot.

prioritized list created by the bidding agent. Lot priority is based on proximity to the bidding agent and proximity to the nest. The bid also includes a completion time estimate, which indicates when the agent would expect to finish the search of that lot.

Success of the bid is based on whether the lot associated with the bid has been previously searched or is currently under contract to any agent. If either of these conditions are true, the bid fails and the bidding agent submits a new bid for the next lot on its prioritized list. One important feature of this process is that the bid can be rejected by any agent with knowledge that the bid lot either has already been searched or is under contract. This decentralizes the contracting process and makes it tolerant of agent failures since no single agent is responsible for determining

whether a bid succeeds.

If the bid is not rejected, the agent is assumed to have contracted to search the lot. Once the agent has the contract, it proceeds to the contracted lot and commences a search of the lot. When the lot has been searched, the agent broadcasts a *search complete* so that other agents will know the lot has been searched. The agent then compiles a new prioritized list of lots and starts the bidding process once again.

The fact that agents only contract for one lot at a time is another key component of the robustness of Hopscotch, because an agent that fails only delays completion of that one lot. Once the contract for that lot expires, the next new bid for it will be successful and a different agent will search the lot.

Tracking of contracts and completed lots is the responsibility of all agents collectively. Contracts are added to a list of in-work contracts once a bid is accepted. When a search contract is reported complete, the contract is removed from the list of in-work contracts and the lot is added to a list of searched lots. Presence of a lot in either list results in rejection of any new bid for that lot. Any time a new bid is received, the list of in-work contracts is reviewed for expired contracts prior to evaluation of the new bid. A contract is expired when it remains on the in-work contract list past the estimated completion time, meaning that a search complete message associated with the contract was not received by the time the contracting agent was supposed to have completed the search. When this happens the contracting agent is assumed to have failed. The contract is removed from the in-work contracts list and thus the associated lot is made available for subsequent bids from any agent. This approach gives the strategy robustness against agent failures.

One of the advantages of the Hopscotch strategy is the simplicity of the communications required. While two-way communication between agents is required, communication is infrequent, can be asynchronous, and the messages exchanged

are short and simple. Gerkey and Matarić [48] point out numerous advantages to using simple asynchronous communications in their implementation of an auction-based multi-robot task management system. Howard et al. [36] discuss the problems encountered using WiFi communication in their multi-robot implementation; their experience suggests the great benefit of keeping communications simple.

With the Hopscotch strategy, messages are only exchanged when an agent is between contracts, and all that is required is transmission of a bid by a contracting agent, a response to the bid from other agents, and a notification to other agents from the contracting agent when the contract has been fulfilled. Furthermore, a response to a bid is only required in the case of rejection, since the absence of a rejection results in the contracting agent assuming responsibility for execution of the contract. The message size required varies with specific implementation, but the data content of each message is on the order of tens of bytes.

The Hopscotch strategy does not necessarily require global communication. Since maintaining a list of searched lots is a collective responsibility, each agent maintains a list of lots it knows to have been searched based on the agent's own work and responses to the agent's bids. Because lot bidding priority is biased to give higher priority to nearby lots, agents in a localized area are likely to have knowledge of local lots, enabling them to respond to bids from nearby agents. This means that the Hopscotch strategy can function when communication range is limited. Chapter 3 shows results that demonstrate good performance with limited communication range.

2.3 Failure Rates

An agent's chance of failure at each step in the simulation is based on the failure rate. A typical expression of failure rate is Mean Time To Failure (MTTF), which is

expresses the mean useful life of a machine. This more accurately expresses failure rates for machines that are out of reach and thus cannot be repaired, since the more common term Mean Time Between Failures (MTBF) generally assumes the machine will fail, be repaired, and returned to operation multiple times over its lifetime [49]. Failures can be inherent to the machine itself or due to external environmental factors. Both components are included in the overall failure rate.

The machine-inherent failure data is derived from both testing and analysis of the components that make up the machine. The classical representation of failures over time is a "bathtub" shaped curve, where there is a high chance of failure initially due to "infant mortality", dropping to a constant failure rate as components have "burned in", and then rising again as components reach the end of their design life and wear out. A notional MTTF curve is shown in figure 2.4.

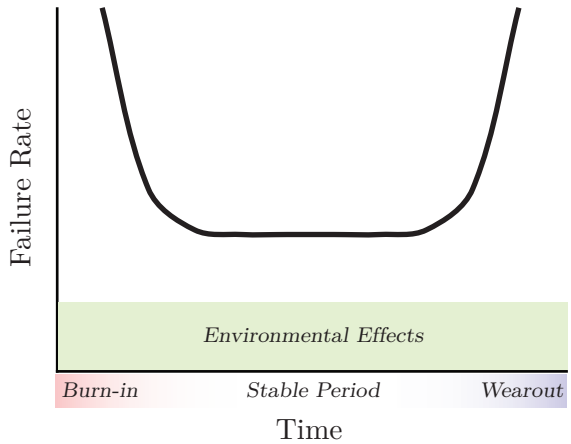


Figure 2.4: Typical failure rate over the life of a machine. The failure rate due to environmental effects is added to the machine's inherent failure rate [49].

When a machine is in an unknown environment, and especially when the machine is mobile, the machine may encounter hazards that lead to failures. The overall failure rate is the sum of the failure rate of the machine itself and failure due to the environment [50]. While there may be certain areas of the space that are more haz-

ardous than others, and certain times during the exploration when the risk of damage is greater, one of the suppositions here is that the exact nature of the environment is unknown and therefore can only be generally assumed, so a constant is a fair way to represent this component of the overall failure rate.

For these experiments, it is stipulated that the exploration occurs over the flat, constant region of the failure rate curve. When considering how robot agents would be used, it is reasonable to expect that before being sent into an environment where they cannot be reached, they would have gone through a burn-in period. Furthermore, it would be unlikely that machines that were approaching the end of their expected life would be used for critical tasks. In addition, the exploration time considered will be small relative to the design life of the machines. For these reasons, the agent failure rate is modeled here as a constant.

2.4 Metrics

In [21] Balch and Arkin introduce three metrics for multi-agent tasks, two of which are adapted for use in this work. The first is *time*, a measure of how long it takes to perform a task, with the goal of minimizing that time. Here, time refers to the elapsed time to achieve some specified fraction of search completeness. The second is *energy*, where the goal is to complete the task using the smallest amount of energy. The energy metric is translated to the more informational metric of *redundancy* as described below. The third they term *reliability/survivability*, the goal of which is to measure the probability of task completion at the expense of any time or cost. While *reliability/survivability* is not directly applicable here, *robustness*, which is closely related, is evaluated using measurements of time and energy to compare performance given a particular agent failure rate.

To formally define the time metric, let $s(l, t)$ represent the number of times a

location l has been searched over all time steps $1 \dots t$. Because the search goal is not always complete coverage, the time metric is associated with a search coverage fraction c. The time metric therefore refers to the time t at which the number of locations in S where $s(l,t) > 0$ divided by the total number of locations in S reaches c.

As previously described, operating agents are almost always searching and moving simultaneously. Therefore, a searching agent is considered to expend one unit of energy per time step, so total energy is simply the number of searching agents integrated over time. It is more insightful to consider energy spent at each location, which indicates the amount of *redundant* search that has occurred. Since by definition a location only needs to be visited once to be considered searched, multiple visits use excess energy and are considered redundant. The count of locations that have been redundantly searched (i.e., locations where $s(l,t) > 1, s(l,t) > 2, \dots$) will be used as a proxy for the energy metric.

Using these metrics, the effectiveness of the various search strategies will be evaluated. As agents fail during the runs, these two metrics provide a measure of how well each strategy copes with the failures. The importance of each respective metric depends on the application, but in general the goal is to minimize time and redundancy.

Chapter 3

Results

3.1 Time and Redundancy

Figure 3.1 shows the time and redundancy performance of various strategies. The data come from runs using a 200×200 search space with the nest in the center. Each experimental run started with 20 agents, each agent having a 10^{-4} failure rate, run over 10^4 time steps. Search data shown is averaged over 16 runs for each strategy. The top black lines indicate the fraction of locations searched (given on the left y axis) over time. The shaded regions below the black lines indicate the number of times that fraction of locations was searched; from dark green to bright green, the locations were searched one to 10+ times, respectively. The dark red line indicates the number of active agents at a given time in the run, given on the right y axis. The standard deviation from the average search progress over the 16 runs is shown by the orange dashed lines.

As agents fail over the course of the run, the rate of progress is certainly affected. However, the primary influence on the rate of progress for the non-coordinated algorithms in (a), (b), and (e) seems to be space remaining. This is what would be

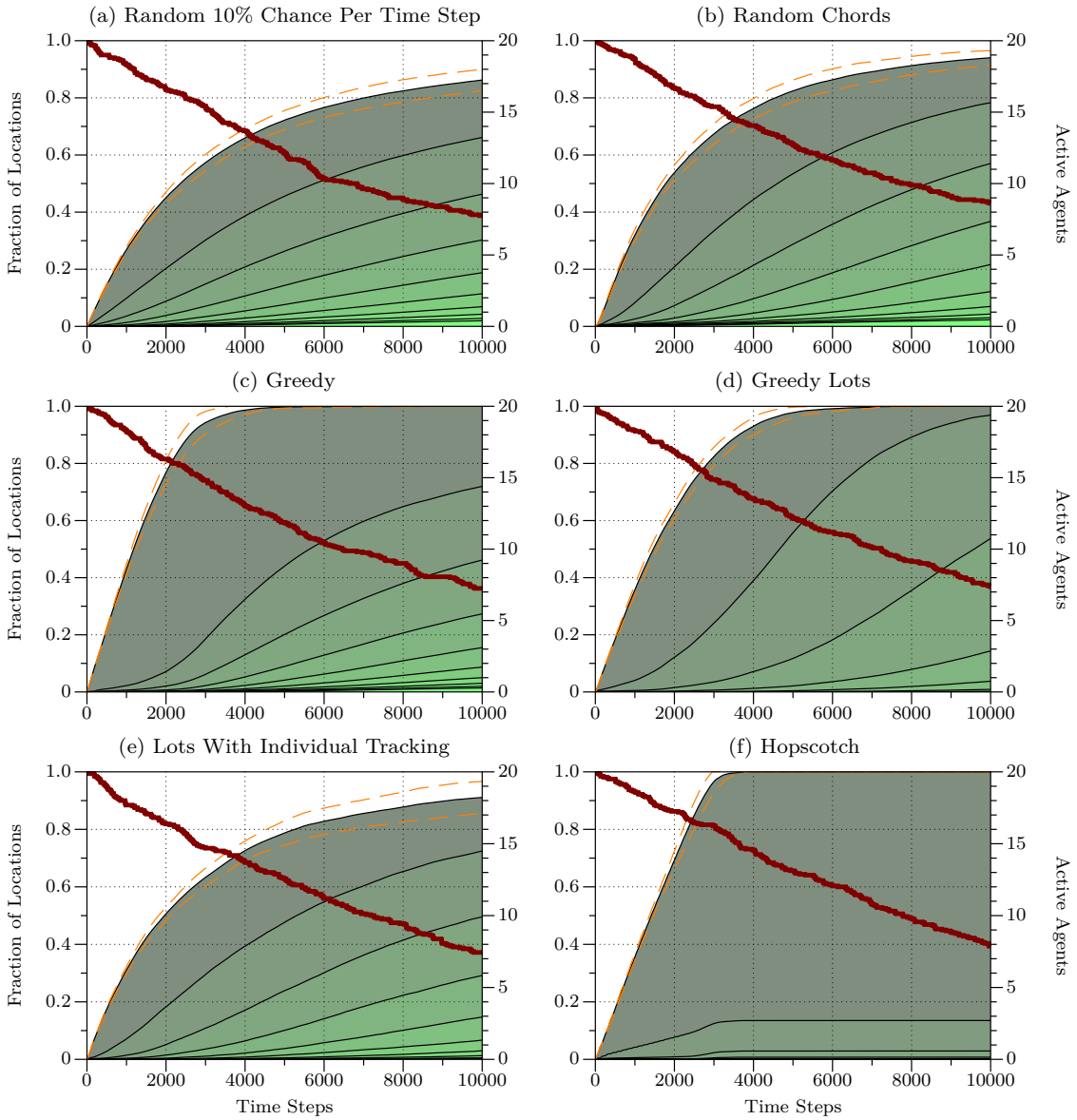


Figure 3.1: Space searched over time using different search strategies. The shaded green regions indicate the number of times that fraction of locations was searched; from dark green to bright green, the locations were searched one to 10+ times, respectively. These colors match the key for figure 2.1. Dashed orange lines show plus and minus one standard deviation, respectively. The dark red lines indicate the number of active agents at a given time in the run, with values given on the right y axis.

intuitively expected, since for (a) and (b) there is nothing influencing the agents to go to locations not previously searched, and for (e) the agents do not know which locations have been searched by other agents.

The brighter green colored regions indicate redundancy, since search has been defined as requiring only one visit to each location. As the number of locations that have not been searched decreases, agents will search areas that have already been searched, resulting in multiple searches at a larger number of locations. Search efficiency is discussed further below.

The two random search strategies, shown in plots (a) and (b), yield similarly shaped progress curves. As expected, the fraction of locations searched becomes asymptotic to 1 as locations that have not been searched become scarce. Also, the better performance of Random Chords strategy as compared to the Random Direction Change strategy is apparent, with the probable reason being that turning is much less frequent.

The Greedy strategy (c) starts out with excellent performance. Eventually, though, some agents end up being "painted in" (having no neighboring locations that have not been searched) before the search completes and revert to the less effective random search strategy in an attempt to find locations that have not been searched. Still, for the cost of implementing the simple communication system required for this strategy, performance is achieved that is significantly better than any of the non-coordinated strategies.

The 16 run average for Lots with Individual Tracking shown in plot (e) is smooth, but plots of individual runs (not shown) indicate choppy progress, which is to be expected when the search space is decomposed into lots that are not searched in a coordinated manner. An agent searching a lot that has already been searched by another agent spends that chunk of time contributing nothing; however, an agent

searching a lot that has not been searched is extremely efficient while searching that lot.

One remarkable feature of the Greedy Lots plot (d) is the wide spreading of the second and third regions (indicating locations searched twice and three times, respectively) below the darkest (searched once) region. The Greedy Lots strategy is the only strategy of the six that considers the number of times searched in choosing where to search; the other non-random strategies consider locations in a binary manner as either searched or not searched. The characteristic of considering times searched, and maintaining a smooth buildup of times searched across the space, lends the strategy to use in a situation where agents have imperfect sensors or undetected sensor failures. In this situation, the second search is more valuable than the third search, and so on, with regard to overall detection [51].

The Hopscotch strategy (f) achieves excellent performance with very little redundancy. What inefficiency there is is primarily due to the necessity of traversing searched lots to get to a lot that has not been searched. If there are an average of 17 active agents over the first 2.5×10^3 time steps, dividing 17 agents into 4.0×10^4 locations gives a best case (given an ideal strategy and the expected agent failure rate) search time of approximately 2.4×10^3 time steps. The performance shown is approximately 1.5 times this best case performance.

Also notable is the characteristic of the Hopscotch strategy that all agent activity stops once the search completes. This is shown on graph (f) by all of the progress lines going to a slope of zero when the search is complete. This behavior is not seen with the other strategies, since Hopscotch is the only strategy where agents are aware when the space has been completely searched.

Figure 3.2 provides further insight into how much redundant search is taking place. These data are based on the same run parameters as previously described,

averaged over 16 runs. This chart contrasts the non-decomposed strategies with the lot-based strategies. The data for Random Direction Change, Random Chords, and Greedy strategies show a fraction of locations searched over 32 times; this is primarily due to effects at the boundary of the search space where an agent that is unable to continue will turn and travel along the border locations. The lot-based strategies do show some well-searched travel lanes to lot corners, which share common x or y values, but generally avoid the border effects.

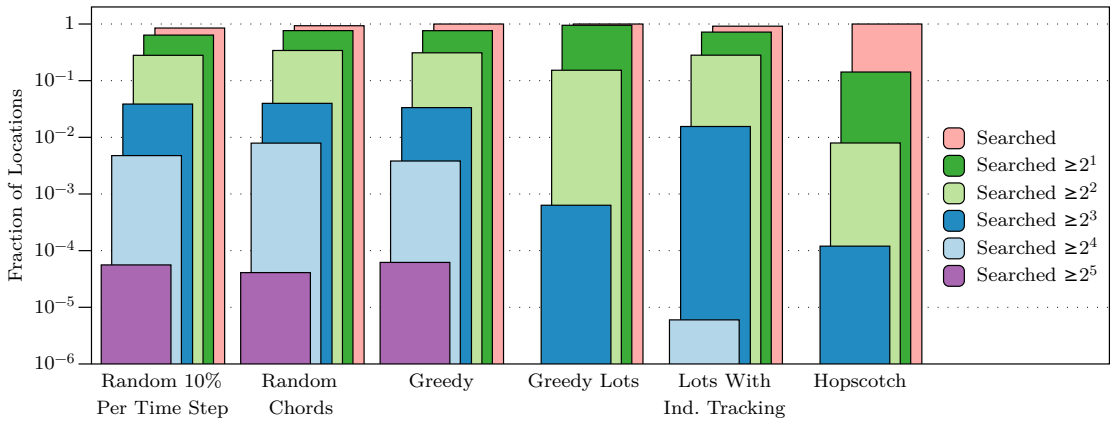


Figure 3.2: Comparison of the fraction of locations redundantly searched for various search strategies.

3.2 Number of Agents

Figure 3.3 shows the effect of starting with different numbers of agents. The run parameters are the same as for the previously shown data, except that the runs began with 10, 20, 30, and 40 agents. As expected, using more agents generally results in better performance. The performance gains, however, are less with each added agent, especially with the uncoordinated strategies. The coordinated strategies, especially the Hopscotch strategy, get the most performance from each added agent.

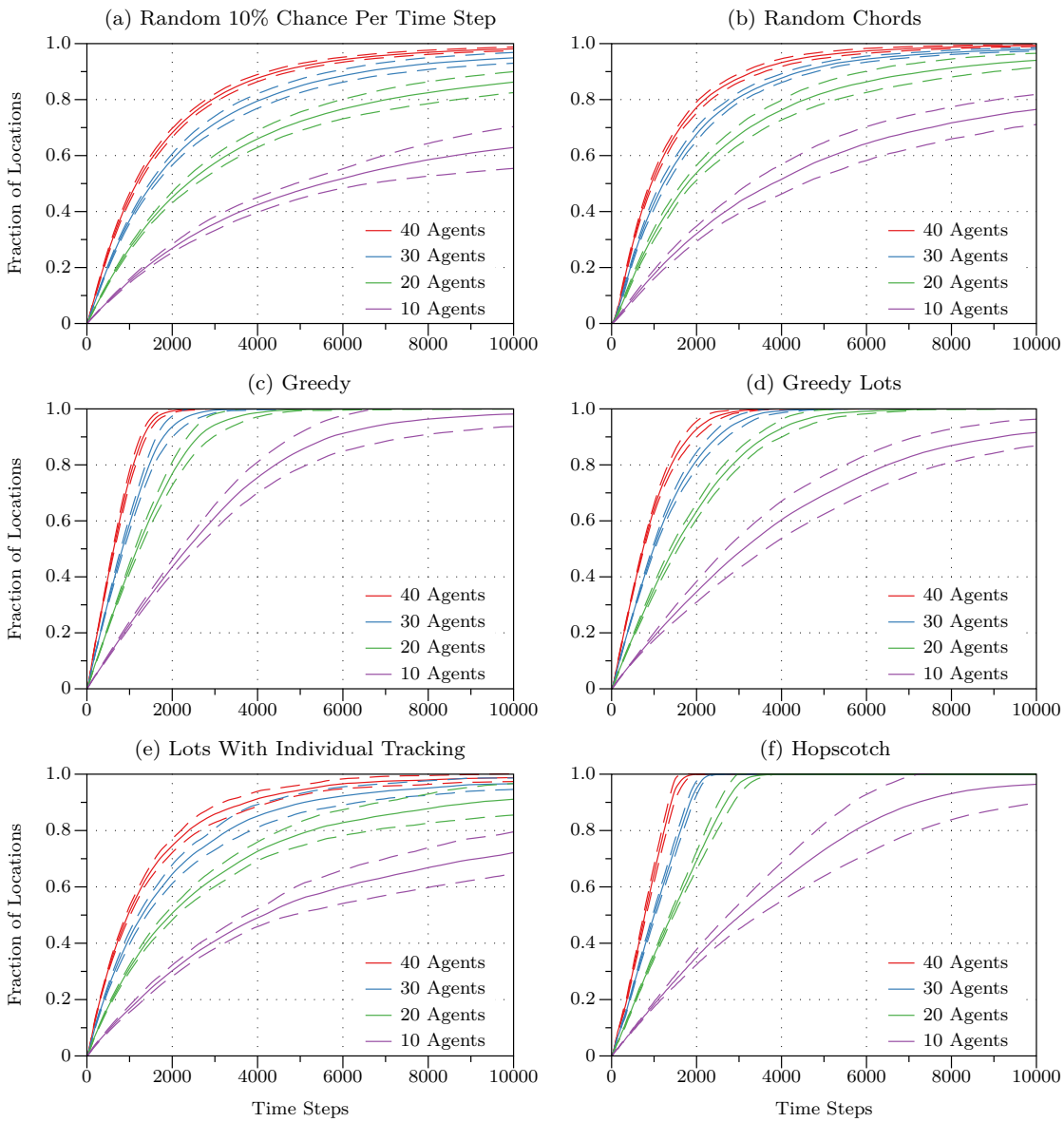


Figure 3.3: Space searched over time with a varying number of agents at the start of the runs. The dashed lines represent plus and minus one standard deviation, respectively.

3.3 Completeness

Figure 3.4 shows a comparison of the time to complete fractions of the search task. Strategies other than Hopscotch do not always complete before all agents fail, so their average never reaches 100%. Because of this, the figure uses 99.5% rather than 100% as the last increment. If a complete search is required, Hopscotch and Lots With Individual Tracking will meet the requirement as long as at least one agent remains operational. However, Lots With Individual Tracking is so much slower that, for these runs, all of the agents failed before Lots With Individual Tracking completed the search. All of the strategies achieved 60% coverage by time step 4000, so if this is sufficient performance, one of the uncoordinated strategies could be used, avoiding the complication of implementing communications.

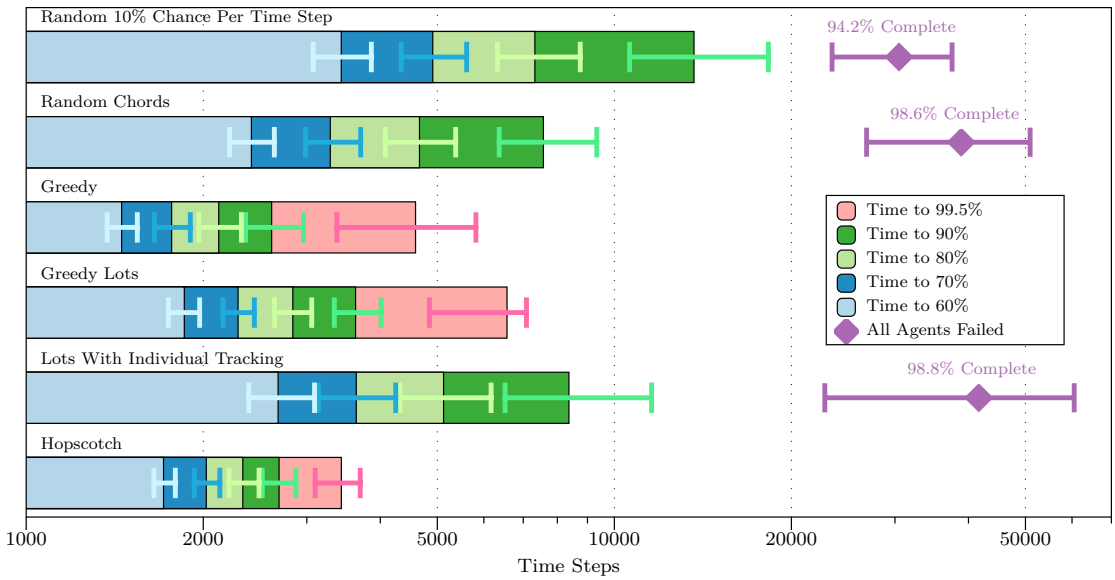


Figure 3.4: Comparison of the time to complete portions of the search task for different strategies, with time to complete $+1/-1$ standard deviation from average search progress. Where all agents failed before the search completed, average time of the final failure is shown by the diamond marker with plus and minus one standard deviation, along with the average completeness at that time.

3.4 Hopscotch Lot Size and Failure Rate

When using the Hopscotch search strategy, the choice of lot size can have a significant impact on performance. Larger lots can be searched more efficiently because on average agents search more locations before they have to turn. However, larger lots also take longer to search, meaning more work could be lost if an agent fails while searching a large lot. Figure 3.5 shows this relationship for three different failure rates. The data come from runs with a 480×480 search space with the nest in the center. 64 agents started each search. Each data point is the average of four runs with the indicated lot size and failure rate.

The plot shows that larger lots improve Hopscotch performance up to a point, but larger lots can also have a detrimental effect on performance with high failure rates. Comparison of 60% and 90% curves shows that the detrimental effect is more pronounced when a higher percent of search coverage is desired, likely a result of

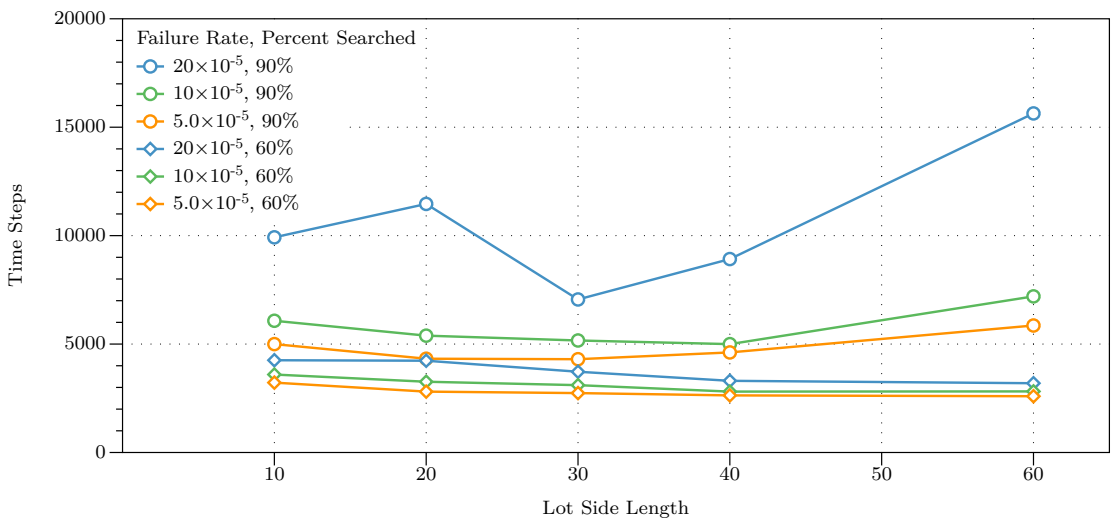


Figure 3.5: Hopscotch search time as a function of lot size and agent failure rate. Curves for time to 60% search completion and 90% completion are shown for each of three different failure rates.

more agents failing due to the longer search duration required.

3.5 Effect on Hopscotch of Limited Communication Range

Figure 3.6 shows a comparison of the time to complete fractions of the search task using a Hopscotch strategy with limited communication range. In order to minimize the effects of space borders on agent concentration, a much larger search space was used. The search space for these runs was 480×480 with the nest in the center. 20 agents started each search, with a 2×10^{-5} failure rate for each agent. Each data point is the average of four runs with the indicated communication and lot size.

The most obvious result from these plots is that increasing communication range has a positive effect on performance. An interesting result shown in these plots is that, except for the small 10×10 lots, there is a great improvement in performance up to a communication range of roughly three times the lot edge length. This indicates that there is definite benefit to considering communication range when specifying lot size, and vice versa.

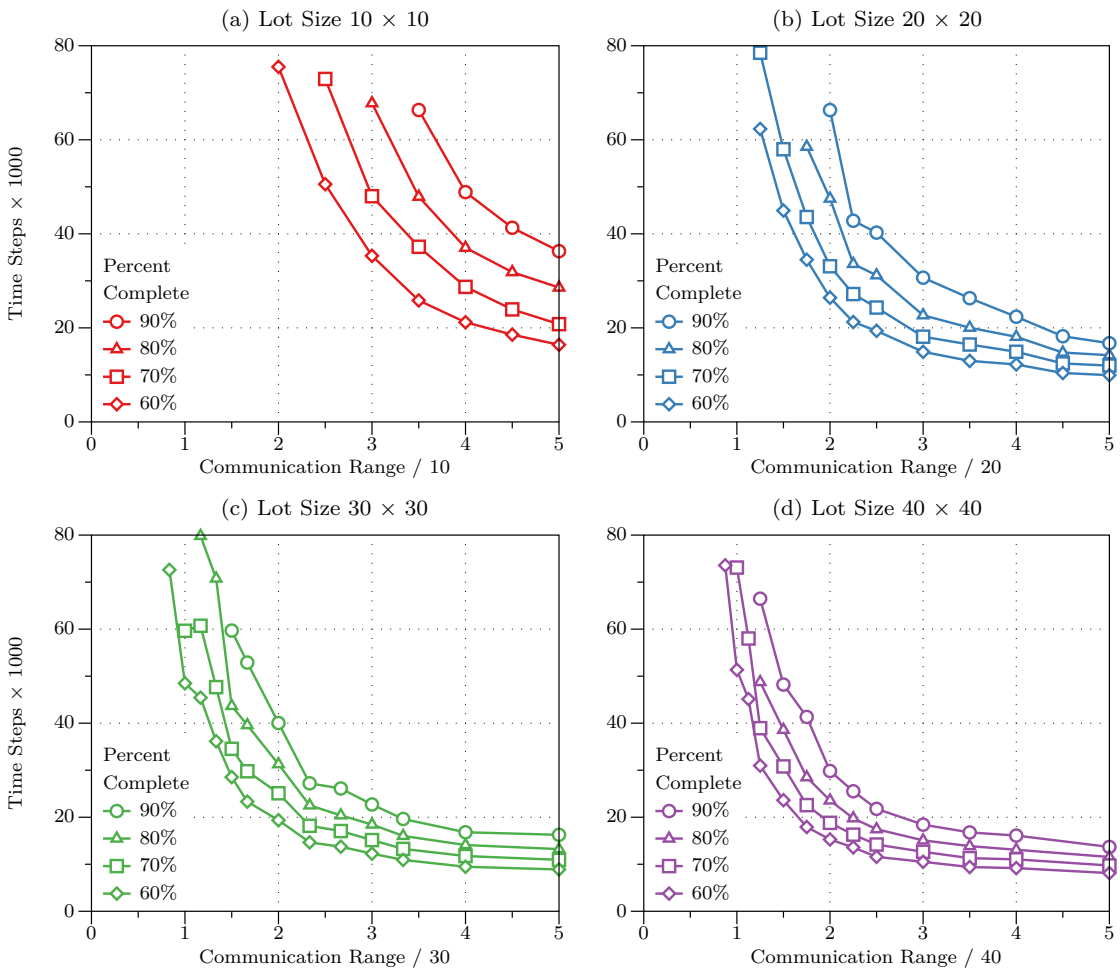


Figure 3.6: Comparison of the effect of communication range on performance with various Hopscotch lot sizes. The x-axis for each graph is in units of communication range divided by the length of a lot side.

Chapter 4

Conclusions

4.1 Summary

The work described in this thesis compared various basic multi-agent search strategies featuring different types of approaches and requirements. Hopscotch, a coordinated strategy designed for robust handling of agent failures, was introduced and characterized. All of the strategies were challenged with agent failures during testing. Of the strategies considered, the Hopscotch strategy had the best performance based on both of the metrics considered here, time and redundancy, but requires implementation of a reliable two-way communication network to function properly. The Greedy and Greedy Lots strategies also achieved impressive time performance but had more redundancy, especially as the search got close to being complete and locations that had not been searched were difficult to greedily find. While the greedy strategies also require communication between agents, this communication can be implemented using a simple pheromone-like marking scheme. The uncoordinated strategies have fairly good time performance when only fractional search coverage is required. Since the agents required to implement the random strategy are not very complicated, it

may be most economical to improve time performance of random strategies simply by increasing the number of agents used.

The six strategies considered implement different ways of dealing with failures, with mixed results. The uncoordinated strategies (Random Direction Change, Random Chords, and Lots With Individual Tracking) have an inherent duplication of effort that tends to hide agent failures. This duplication of effort explains their very poor performance as measured by both the time and redundancy metrics. The greedy strategies' marking system handles agent failures implicitly, without requiring extra consideration. Because of this, the greedy strategies seem be least affected by failure. In contrast, Hopscotch is fully coordinated and therefore very powerful and adaptable, but needs explicit failure handling for best performance. Hopscotch and Greedy performance are very close through the first 80% of the search, showing that the explicit failure handling of the Hopscotch strategy performs on par with the implicit failure handling of the Greedy strategy.

The Hopscotch strategy does well because it is a coordinated strategy with features designed to handle agent failures. Because it is coordinated, work is explicitly divided between agents. This reduces duplicate work, resulting in the minimization of both time and redundancy as shown in the simulation data. The Hopscotch contract net implementation features real-time subtask assignment and decentralized subtask management. Simulation results indicate that these features allow the Hopscotch strategy to continue to work well even after agents fail.

4.2 Future Work

A real-world application may need to deal with unknown, arbitrary spaces and obstacles. The Hopscotch algorithm could potentially be expanded to handle these scenarios. This type of implementation could also handle variances in the search

space such as varying level of hazard, varying search value, or hampered mobility by assigning these attributes to lots when they are discovered and considering them when prioritizing lots and evaluating bids. Obstacles and borders could be detected and tracked along with contracted and completed lots, allowing agents to create a map of the space that could be used during the bidding process.

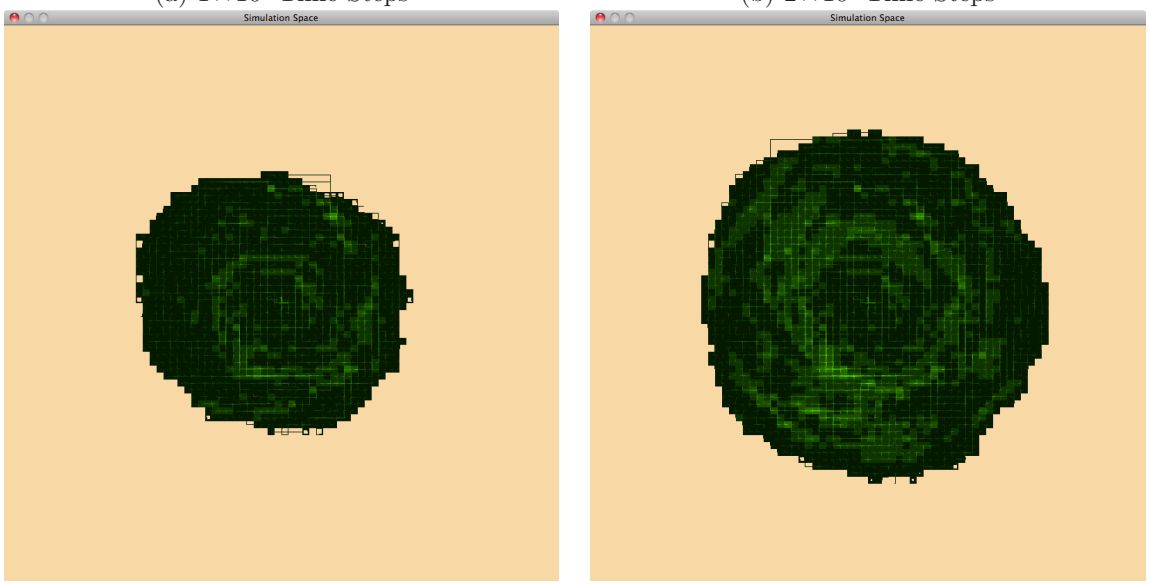
Some applications may require handling other types of real-world problems such as location error and sensor error. Location error can be modeled and compensated for by, for example, adjusting lot size so that lots overlap. If the search sensor is less than perfect, lots could be assigned a "search quality" value rather than the current binary status of searched or not searched, with this search quality used in lot prioritization.

To explore the potential of the Hopscotch strategy, a simulation was run using a pseudo-infinite search space; that is, the space was large enough that the borders were never encountered and thus did not affect agent behavior. To maintain realism, the communication range-limited version of Hopscotch was used with a modest communication range of 60. The ruleset was not modified in any way for this demonstration, and only 20 agents were used with no failures. The search progress as the run progressed is shown in figure 4.1. Because of the nest proximity bias component of lot prioritization, the Hopscotch strategy maintained a contiguous region of searched area centered at the nest. It would be interesting to investigate applications of Hopscotch for searching large or unbounded environments, and how it could be tuned to perform well in these applications.

There are other interesting opportunities to extend the capabilities of the Hopscotch algorithm. The Hopscotch contract net could be used for any complicated space that can be decomposed into lots. It is not necessary that the graph of the lots be fully connected as long as all vertices (lots) are reachable. Spaces with this type of structure includes hotels with rooms connecting to a common corridor, or

(a) 1×10^4 Time Steps

(b) 2×10^4 Time Steps



(c) 3×10^4 Time Steps

(d) 4×10^4 Time Steps

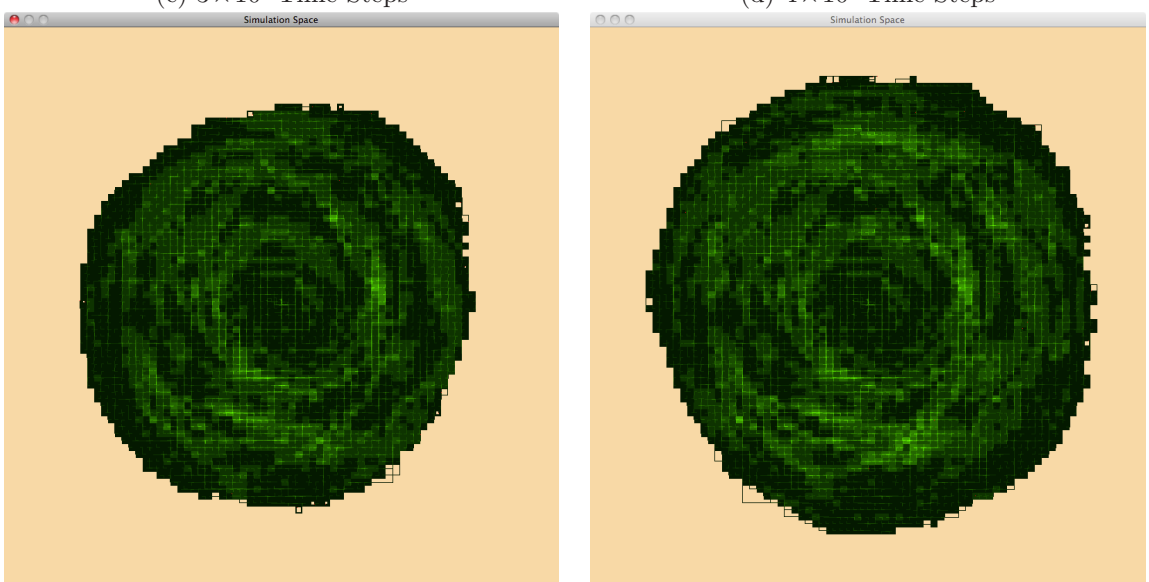


Figure 4.1: Progress of Hopscotch search of a very large space. A color key is given in figure 2.1.

an office where the space is weakly divided into cubicles. For these cases, though, pure travel time could become a significant factor, so there may be opportunities to improve the algorithm using some of the graph-based optimization techniques [29] or roadmap techniques [52, 30] discussed in the literature.

In non-uniform search spaces, or in situations where there are specific search targets that have a non-uniform distribution, the decomposition of the search space into lots provides advantages that the Hopscotch strategy can take advantage of. For example, a lot with multiple expired contracts could be flagged as hazardous, with appropriate measures taken to protect agents. When there are specific search targets that are likely to be clustered, for example fossils or a species of plant, search results for one lot could be used to bias the priority of nearby lots.

One interesting result of these experiments was the excellent time performance of the greedy strategies. There may be ways of incorporating greedy elements into the Hopscotch strategy to improve performance further. The marking techniques used for the greedy strategies could also provide an effective fallback capability in cases where two-way communication is unavailable.

Finally, although agent failure is considered here to be a binary state, there are various types of failures (e.g., sensor, navigation, communication) that may leave agents partially operational, or even malevolent. It may be possible to identify these types of failures based on agent behavior and adjust the search rules to compensate for them.

References

[1] Pedro Santana, Jose Barata, Hildebrando Cruz, Antonio Mestre, Joao Lisboa, and Luís Flores. A multi-robot system for landmine detection. *Emerging Technologies and Factory Automation*, 2005. ETFA 2005. 10th IEEE Conference on. Vol. 1. IEEE, 2005.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on.* Vol. 1. IEEE, 2000.

[3] Terry Huntsberger, Paolo Pirjanian, Ashitey Trebi-Ollennu, Hari Das Nayar, Hrand Aghazarian, Anthony J. Ganino, Mike Garrett, Sanjay S. Joshi, and Paul S. Schenker. CAMPOUT: A control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 33.5 (2003): 550-559.

[4] Yaniv Altshuler, Vladimir Yanovsky, Israel A. Wagner, and Alfred M. Bruckstein. Swarm intelligence– searchers, cleaners and hunters. *Swarm Intelligent Systems* (2006): 93-132.

[5] R. Beckers, S. Goss, Jean-Louis Deneubourg, and J. M. Pasteels. Colony size, communication, and ant foraging strategy. *Psyche* 96.3-4 (1989): 239-256.

[6] Donald E. Franklin, Andrew B. Kahng, and M. Anthony Lewis. Distributed sensing and probing with multiple search agents: toward system-level landmine detection solutions. *SPIE's 1995 Symposium on OE/Aerospace Sensing and Dual Use Photonics.* International Society for Optics and Photonics, 1995.

[7] David Jung, Gordon Cheng, and Alexander Zelinsky. Robot cleaning: An application of distributed planning and real-time vision. *International conference on Field and Service Robotics.* 1997.

[8] Chaomin Luo and Simon X. Yang. A real-time cooperative sweeping strategy for multiple cleaning robots. *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on.* IEEE, 2002.

[9] Howie Choset. Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31.1 (2001): 113-126.

[10] Stephen B. Stancliff, John Dolan, and Ashitey Trebi-Ollennu. Planning to fail–Reliability needs to be considered a priori in multirobot task allocation. *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on.* IEEE, 2009.

[11] Gregory Dudek, Michael RM Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots* 3.4 (1996): 375-397.

[12] Marco Dorigo, Marco, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, eds. *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings.* Vol. 4150. Springer, 2006.

[13] T. Flanagan, K. Letendre, and M. E. Moses. Quantifying the Effect of Colony Size and Food Distribution on Harvester Ant Foraging. *PloS one* 7.7 (2012): e39427.

[14] S. Verret. Current state of the art in multirobot systems. *Defence Research and Development Canada-Suffield, Ralston ALTA, Technical Memorandum DRDC-SUFFIELD-TM-2005-241* (2005).

[15] Y. Uny Cao, Alex S. Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots* 4.1 (1997): 7-27.

[16] Curt Berenton and Pradeep Khosla. An analysis of cooperative repair capabilities in a team of robots. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on.* Vol. 1. IEEE, 2002.

[17] Wooram Park, David Albright, Charles Eddleston, Wai K. Won, Kiju Lee, G.S. Chirikjian. Robotic Self-Repair in a Semi-Structured Environment. *Workshop Proceedings of Self-Sustaining Robotic Systems.* 2004.

[18] Fredric Chenavier and James L. Crowley. Position estimation for a mobile robot using vision and odometry. *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on.* IEEE, 1992.

[19] X. P. Yang and S. Y. Liu. Mobile robot locating and tracking system design based on wireless sensor network. *Chinese Journal of Electron Devices* 30.6 (2007): 2265-2268.

[20] Ercan U. Acar, Howie Choset, Yangang Zhang and Mark Schervish. Path Planning for Robotic Demining: Robust Sensor-Based Coverage of Unstructured Environments and Probabilistic Methods. *The International Journal of Robotics Research* 22.7-8 (2003): 441-466.

[21] Tucker Balch and Ronald C. Arkin. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots* 1.1 (1994): 27-52.

[22] Douglas W. Gage. Randomized search strategies with imperfect sensors. *Proceedings of SPIE.* Vol. 2058. 1994.

[23] Sven Koenig, Boleslaw Szymanski, and Yaxin Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence* 31.1 (2001): 41-76.

[24] Sven Koenig and Yaxin Liu. Terrain coverage with ant robots: a simulation study. *Proceedings of the fifth international conference on Autonomous agents.* ACM, 2001.

[25] I. Wagner and A. Bruckstein. Cooperative cleaners: a study in ant robotics. Technical Report CIS9512, Technion, 1995.

[26] Eliyahu Osherovich, Vladimir Yanovki, Israel A. Wagner and Alfred M. Bruckstein. Robust and Efficient Covering of Unknown Continuous Domains with Simple, Ant-Like A(ge)nts. *The International Journal of Robotics Research* 27.7 (2008): 815-831.

[27] Bijan Ranjbar-Sahraei, Gerhard Weiss, and Ali Nakisaee. Stigmergic coverage algorithm for multi-robot systems. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3.* International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[28] Ani M. Hsieh, Ádám Halász, Spring Berman, and Vikay Kumar. Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence* 2.2 (2008): 121-141.

[29] Noam Hazon and Gal A. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems* 56.12 (2008): 1102-1114.

[30] O. Bayazit, Jyh-Ming Lien, and Nancy Amato. Better group behaviors using rule-based roadmaps. *Algorithmic Foundations of Robotics V* (2004): 95-112.

[31] Wolfram Burgard, Mark Moors, and Frank Schneider. Collaborative Exploration of Unknown Environments with Teams of Mobile Robots. *Advances in plan-based control of robotic agents* (2002): 187-215.

[32] Rajeev Alur, Aveek J. Das, Joel Esposito, Rafael Fierro, Gregory Grudic, Yerang Hur, R. Vijay Kumar, Insup Lee, James Ostrowski, George J. Pappas, B. Southall, John R. Spletzer, and Camillo J. Taylor. A framework and architecture for multirobot coordination. *Experimental Robotics VII* (2001): 303-312.

[33] Nicola Bezzo and Rafael Fierro. Swarming of mobile router networks. *American Control Conference (ACC)*, 2011. IEEE, 2011.

[34] Jorge Cortés, Sonia Martínez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *Robotics and Automation, IEEE Transactions on* 20.2 (2004): 243-255.

[35] Francesco Bullo, Ruggero Carli, and Paolo Frasca. Gossip coverage control for robotic networks: dynamical systems on the space of partitions. *SIAM Journal on Control and Optimization* 50.1 (2012): 419-447

[36] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research* 25.5-6 (2006): 431-447.

[37] Spring Berman, Adam Halasz, M. Ani Hsieh, and Vijay Kumar. Optimized stochastic policies for task allocation in swarms of robots. *Robotics, IEEE Transactions on* 25.4 (2009): 927-937.

[38] Reid G. Smith, The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on* 100.12 (1980): 1104-1113.

[39] Han-Lim Choi, Luc Brunet, and Jonathan P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *Robotics, IEEE Transactions on* 25.4 (2009): 912-926.

[40] Sanem Sariel and Tucker Balch. Real Time Auction Based Allocation of Tasks for Multi-Robot Exploration Problem in Dynamic Environments. *Proceedings of the AAAI-05 Workshop on Integrating Planning into Scheduling.* 2005.

[41] S. B. Stancliff, John M. Dolan, and A. Trebi-Ollennu. Mission reliability estimation for multirobot team design. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006.

[42] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A Low Cost Scalable Robot System for Collective Behaviors. *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012.

[43] *The Objective-C 2.0 Programming Language.* Apple Inc. 2009.

[44] Weisstein, Eric W. von Neumann Neighborhood. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/vonNeumannNeighborhood.html retrieved 10/17/2012.

[45] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Autonomous Robots* 11.3 (2001): 319-324.

[46] Michael J. McNish. Effects of uniform target density on random search. Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1987.

[47] M. Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE* 94.7 (2006): 1257-1270.

[48] Brian P. Gerkey and Maja J. Matarić. Sold!: Auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on* 18.5 (2002): 758-768.

[49] Paul A. Tobias, David C. Trindade. *Applied Reliability.* Chapman & Hall/CRC, 2010.

[50] Stephen Stancliff, John M. Dolan, and Ashitey Trebi-Ollennu. Towards a Predictive Model of Robot Reliability. Robotics Institute Paper 197 (2005).

[51] Douglas W. Gage. Many-robot MCM search systems. *Proceedings of the Autonomous Vehicles in Mine Countermeasures Symposium.* Vol. 9. 1995.

[52] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on.* Vol. 1. IEEE, 1996.