2009

Hierarchical task allocation in robotic exploration

John Hawley

Hierarchical Task Allocation in Robotic Exploration
Master of Science Thesis
B. Thomas Golisano College of Computing and Information Science

John Hawley
Rochester Institute of Technology
Department of Computer Science
102 Lomb Memorial Drive
Rochester, New York 14623
USA

March 19, 2009

Signatures

I, John Hawley, submit this thesis in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. It is approved by the committee members below.

John Hawley

Dr. Zack Butler
Committee Chair

Dr. Minseok Kwon
Reader

Dr. Stanisław P. Radziszowski
Observer

Acknowledgements

Thanks to all the family and friends who have been amazingly helpful and supportive in a variety of ways. Without the motivational prodding of my friends, this would have taken a lot longer. Thanks to Dr. Zack Butler for getting me interested in robotics and taking the time to work with me on an independent study and this thesis. Thanks also to my employer and committee for being so patient. Special thanks to Sara and Nicky for proof-reading, RJ, Carl, and Chris for letting me bounce ideas off them, and Tay and various others for formatting and typesetting help. Thanks to all for moral support and best wishes.

Dedicated to Stephen William Hawley

Abstract

Autonomous robotic exploration has long been a topic of interest in robotics research. Robotic exploration promises the ability to explore otherwise unreachable or hostile environments. Autonomous exploration is particularly useful in distant or hostile environments in which real-time communication with a human controller may not be practical, such as deep sea or planetary exploration [16]. In order to more effectively explore a large unknown area, multiple robots may be employed to work cooperatively. While cooperation among multiple robots allows for increased exploration potential, it also entails significantly more complex planning.

This complex planning involves allocation of exploration tasks to the robots participating in the exploration. Task allocation for multi-agent systems has applications in a wide variety of fields, but specifically in robotics, it makes a level of autonomy possible that is difficult to achieve otherwise. Task allocation has been approached in a variety of ways, depending largely on the nature of the tasks considered. Some problems present very specific tasks, allowing task allocation algorithms for them to be very domain-specific. This thesis presents an analysis of various task allocation approaches that have been taken specifically for autonomous robotic exploration, and will present a new hierarchical market based approach. This new approach provides agents with a mechanism to form coalitions and to divide a coalition into smaller coalitions. The formation of new coalitions from larger coalitions to pursue multiple avenues of exploration forms an implicit hierarchy of goals as they are discovered.

Contents

List of Figures

Chapter 1

Introduction

Autonomous robotic exploration is a task that presents many problems, and in practice is quite difficult to perform. Significant research in the area of robotic exploration has focused on multi-agent exploration strategies [3, 7, 10, 9, 13, 18, 19]. Such strategies offer the promise of not only more timely exploration, but also more robust systems. Robustness is a valuable attribute, particularly when exploring highly unpredictable or hostile environments. Multi-agent systems, however, require more complex exploration strategies than those used in single-agent systems to be effectively utilized to their full potential. These strategies vary in the level of coordination between agents, ranging from minimal communication [18] to extensive communication and planning between agents [7, 10, 9, 19]. Calculation of an optimal exploration strategy becomes increasingly intractable as the size of the environment being explored and the number of agents involved increase. Even in a known environment, calculating the optimal search pattern for multiple agents is an NP-hard problem [9]. Heuristic task allocation methods are employed to calculate an exploration strategy in a reasonable amount of time.

Task allocation in robotic exploration can either be done explicitly, using a strategy such as the one discussed in [12], or implicitly by simply sharing information between agents [3, 17]. Implicit task allocation involves the sharing of map information, allowing agents to explore frontiers discovered by other agents. The agents do not communicate with each other regarding which frontiers they are exploring, leading to the sub-optimal possibility of multiple agents exploring the same frontier. In explicit task allocation, agents communicate with each other to determine which agent will explore each frontier to ensure that multiple agents do not explore the same fron-

tier needlessly. Implicit task allocation is inherently simpler, but fails to coordinate agents' efforts. This can lead to significant drops in efficiency, particularly as the number of agents increases.

This thesis presents a hierarchical task allocation strategy based on existing coalition forming [12] and market based [8, 9, 15, 19] strategies. The motivation for this type of approach is to attempt to account for synergy between agents during the course of exploration, and to provide agents with a mechanism for exploration even when there are fewer goal points to explore than agents. A market based mechanism is presented by which agents may elect to cooperate with each other. In this way, agents form coalitions if doing so is calculated to be more profitable than not. In order to measure the performance of such task allocation strategies, a simulation environment has been implemented in which lower-level tasks such as localization and mapping are abstracted and ignored by agent implementations, allowing the task allocation process to focus on higher level exploration tasks. The simulation will also serve as a visual aid to provide feedback as to the performance of various task allocation strategies.

Chapter 2

Prior Work

This chapter provides background information regarding autonomous robotic exploration and current approaches to it. This includes a description of the problem of robotic exploration, as well as some auxiliary tasks that are involved. Many of these tasks are abstracted by the simulation platform used for experiments, but others are central to the exploration algorithms tested. Section 2.3 provides a detailed examination of various existing task allocation strategies. These existing strategies are used to derive the rudimentary greedy algorithm that is implemented for comparison purposes (See Section 5.3 for more information).

2.1 Robotic Exploration

The primary task of a robotic exploration algorithm is to provide a robot, or agent, with an order in which to visit points in the environment to minimize the amount of time, or some other resource such as power, required to explore the environment. The exact interpretation of what it means for an environment to be explored varies between applications. In some cases, it may be necessary to fully explore the environment, but in other cases it may be preferable to explore particular features first. In planetary exploration, it is likely that a depth-first autonomous exploration approach would be supplemented by human-specified goals of particular interest. For the purposes of this thesis, exploration is the use of sensors to construct a map of the environment, and that exploration is considered complete when all reachable open space in that environment has been revealed by one or more agents.

Exploration of an environment with complete a priori knowledge of that

environment is essentially the Euclidean Travelling Salesman Problem (TSP) and is therefore NP-hard. In most cases, however, robotic exploration is applied to instances where no information is known about the environment a priori, rendering off-line strategies ineffective, such as using a TSP solver to precompute a path. In these cases, agents must decide where to explore based only on information gathered by previous exploration. In practice, it is difficult even to estimate which avenues of exploration will be the most fruitful based on such limited information. More formally, given a state s of the known map, and a current agent pose p, an exploration strategy must provide a way to calculate the next pose for the agent to move toward. That is, an exploration strategy must provide some function $f : S \times P \rightarrow P$, where S is the set of all possible environment states and P is the set of all possible agent poses. The pose of an agent is the combination of the agent's position and orientation.

Many approaches have been taken to autonomous robotic exploration, including purely reactive algorithms and more deliberate ones. More deliberate approaches generally require the selection of goal points within the environment and a means to determine an order in which to visit them. These goal points are generally a subset of all unexplored points. Since a smaller set is easier to order, smaller subsets are often preferred. One of the most fundamental differences between various exploration strategies is the way in which these goal points are determined. Many modern approaches are based on the use of frontiers as goal points [3, 5, 10, 13, 17]. Experiments performed for this thesis will all use frontiers as goal points. Frontiers are identified as contiguous groups of map cells that represent explored open space adjacent to unexplored space. A goal point is created for each such group, and is located at the arithmetic mean of all points in the group. Other approaches to goal identification that have been implemented include random point selection, greedy exploration, and quad-tree subdivision [19]. Random point selection methods simply select random points within unexplored territory to use as goal points. In greedy selection, goal points are chosen as the centers of the largest unexplored regions at any given time. In quad-tree subdivision based goal point selection, the unknown regions of the map are represented using a quad-tree, and the goal points are chosen as the centers of the quad-tree leaf regions. Quad-tree based selection also conveniently provides a measure of the utility of goal points, which may be used to determine the order in which to visit them.

Various approaches have also been taken to ordering those goal points. Some rudimentary but successful approaches simply direct an agent towards the nearest goal point [17]. In the case where multiple agents are partici-

pating in the exploration, however, this task becomes far more complex. In such cases, more advanced techniques are often employed, including market based allocation mechanisms [9, 19], greedy mechanisms [3, 13], and genetic algorithms [10].

2.2 Map Representations

Various map representations have been used in robotics, each with respective advantages and disadvantages for particular tasks. For exploration, the primary tasks involved are map construction and path planning. Most robotic exploration is performed using range based sensors, such as sonar or laser systems, but may also be performed using vision based systems. When using range based sensors for map construction, evidence grids are a convenient map representation because they are inherently conducive to the incorporation of sensor uncertainty. See Figure 2.1 for an example of an evidence grid. An evidence grid map representation is the decomposition of map information into a grid of equally sized cells, where each cell stores a value that represents the probability that that cell is occupied by an obstacle.

Path planning, however, is more easily performed on topological maps than evidence grids. A topological map represents open spaces and the connectivity between them, but usually does not maintain specific information about obstacles in the area. See Figure 2.2 for an example of a topological map. In some cases, topological maps are constructed using an intermediate evidence grid that is based on sensor data. Map data is often first used to construct the Configuration Space, or C-Space, of the agent. The C-Space of an agent is the space containing all possible positions and orientations for the agent.

In most two dimensional simulations, agents have x and y coordinates and an angular heading, for a total of three degrees of freedom. This means the C-Space has three dimensions, but in many cases, an agent is treated as circular by inscribing it in a circle and disregarding heading, so that the C-Space can be treated as two dimensional. Once the C-Space has been constructed, path planning can be performed either by directly using a grid-based representation or by constructing a topological representation of the C-Space first.

Path planning is generally done using a standard A* or D* [14] algorithm. In some implementations, a slightly modified version is used to accommodate the need for costs of simultaneous paths to multiple goals. With evidence

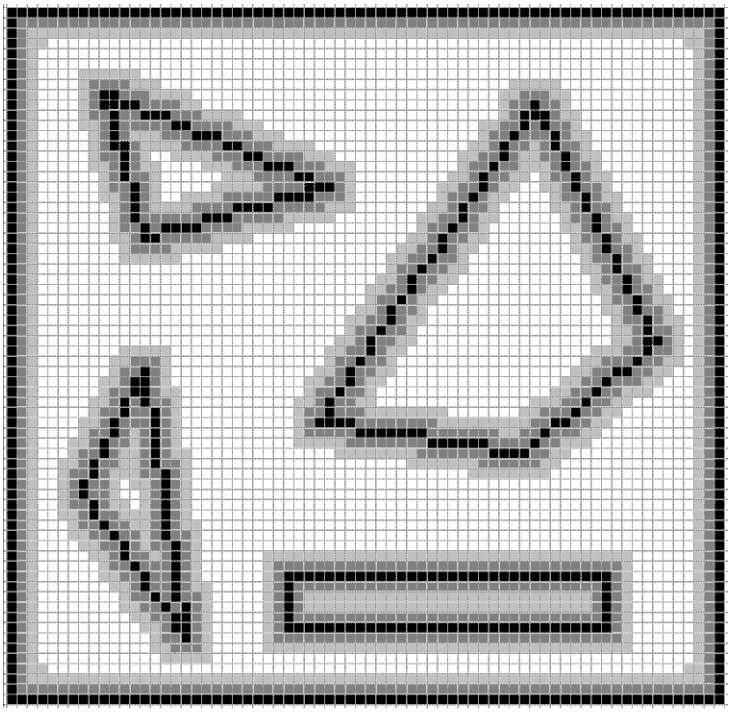

Figure 2.1: An example of an evidence grid map representation. Darker colors indicate increased probability of a cell being occupied by an obstacle.

grids, path planning is performed by treating grid cells as reachable by adjacent open grid cells and performing a search on the resulting search space. When a topological map is used, however, the topological map is used as the search space. In addition, paths to and from the topological map nodes must be calculated in the same way as evidence grid paths, but are generally much shorter and faster to compute. In this way, path planning using topological maps is often significantly faster than with evidence grids alone, depending on the complexity of the topology of the environment and the resolution of the evidence grids used.

2.3 Task Allocation

The most significant difference between single agent exploration and multiple agent exploration is the increased complexity of the task allocation

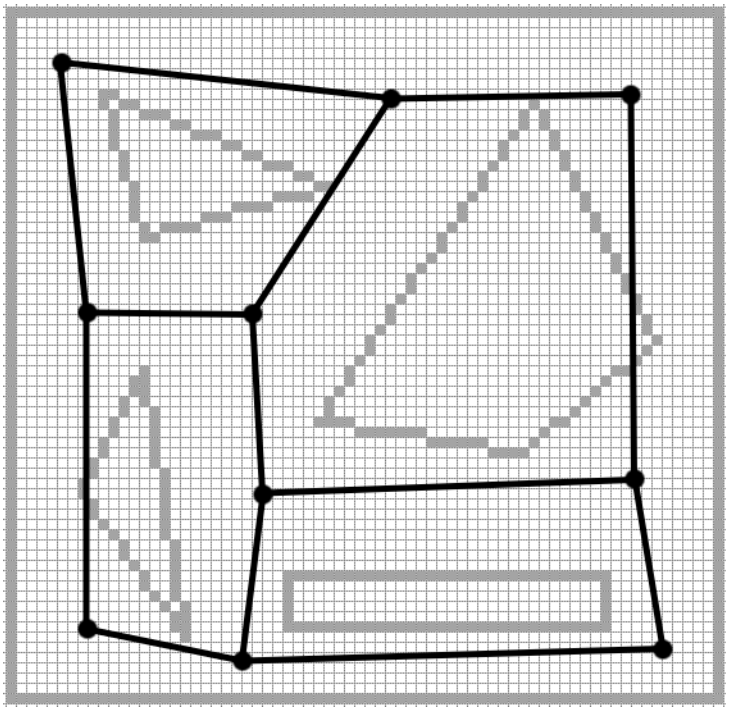

Figure 2.2: An example of a topological map representation. The corresponding evidence grid is shown in grey for clarity, but this information would normally not be incorporated into the topological map itself.

process. Deciding which goal for a single agent to pursue, while difficult, is primarily about choosing an effective heuristic to estimate the utility of each goal. When multiple agents are participating, even if an effective heuristic is chosen, the problem remains that an assignment of agents to goals must also be chosen. Since it may be advantageous, indeed even necessary if there are fewer goals than agents, for multiple agents to pursue the same goal, there are $\|G\|^{\|A\|}$ possible assignments, where A is the set of participating agents and G is the set of goals that need to be explored at a given time. Clearly, an exhaustive approach becomes rapidly intractable as the number of agents and goals increase.

Various alternative approaches have been taken. One of the most naïve approaches is simply to assign each agent to the nearest goal [18]. This approach has the advantages of being extremely simple and completely de-

centralized, with very little messaging overhead. Most recent approaches involve more explicit communication between agents in an attempt to avoid the most suboptimal assignments in which multiple agents are unnecessarily assigned to the same goal. More recent research has focused on more involved approaches. One aspect most of these approaches have in common is the need to evaluate goal points. In order to do this, various measures of *cost* and *utility* have been explored. *Cost* is generally calculated as the distance an agent must travel to reach a goal point, but may be calculated as the time required if all agents do not travel at the same speed. Since the environment is only partially known, the cost is often optimistically calculated by treating unexplored space as open space. For more details regarding *utility* calculation, see Section 2.3.4.

2.3.1 Greedy

Perhaps one of the simplest forms of explicit goal allocation is to greedily choose the most advantageous pairing between a single agent and a single goal at a time. The utility of goals may be adjusted with each successive assignment [3]. In order to perform this kind of adjustment, some mechanism must be used to correlate the utilities of nearby goals. In [3], the authors introduce "expected visibility range" for this purpose. By keeping track of how often a given distance to an obstacle was measured by any agent's sensors, a probability of measuring any given distance can be calculated. This probability is used to discount the utility of nearby frontiers when an assignment is made (See Section 2.3.4 for more details). A similar approach by Simmons et al. in [13] calculates an "expected information gain" region for each goal, which is used to approximate the utility of each goal. The "expected information gain" is calculated by estimating the unexplored area that would be revealed by visiting a frontier and calculating a bounding rectangle for that area. In addition, the overlap of these rectangles is used to discount remaining goals when a goal is assigned to an agent in a way similar to [3]. Further details regarding these utility calculation methods can be found in Section 2.3.4.

In simulations, greedy approaches perform better than the naïve approach where each agent individually pursues the nearest goal without communicating with other agents. Most research regarding greedy allocation mechanisms, however, does not directly address coalition formation, and limits agent cooperation to agents coincidentally pursuing nearby goals. In addition, many greedy implementations, while decentralized, select a single agent to perform the assignment process for any given time interval. Their

decentralized nature makes these systems more robust, but does not necessarily make them scalable to large numbers of agents due to the assignment of all agents when an assignment is calculated. Greedy approaches could be made more scalable by only including agents within reasonable proximity of each other in the greedy task allocation process.

2.3.2 Genetic Algorithm

Another recent approach to frontier allocation is to apply a genetic algorithm to the task [10]. This is a particularly interesting approach, as genetic algorithms are generally unpredictable in terms of computation time. The inspiration to try a genetic algorithm came from the embarrassingly parallel nature and high-dimensional search space abilities of genetic algorithms.

The implementation uses utility and cost calculations very similar to [3]. That is, the utility of each goal is initialized to 1 and is reduced based on the "expected visibility range" and the distance to other nearby goals. Unlike the approach presented in [3], however, the genetic algorithm approach is not iterative. That is, it simultaneously calculates a goal assignment for each agent, and therefore does not discount the utility of nearby goals when one is selected. Instead, it is assumed that the set of agents is homogeneous and therefore all agents have the same visibility range, allowing the utility of each goal to be discounted based solely on the "expected visibility range" and location of other goals. That is, the utility of each goal t is given by

$$U_t = 1 - \left(\sum_{i \in (G - \{t\})} P(d_i) \right)$$

where G is the set of all goals, d_i is the distance from goal t to goal i, and $P(d_i)$ is the probability that the goal cell t can be seen by an agent at goal cell i. Cost is calculated for each agent to all goals by a standard flood-fill algorithm in which a breadth-first search is performed beginning at each agent and continuing until there is a cost known to all goals.

For the purposes of the genetic algorithm, fitness is given by

$$\textit{fitness} = \textit{utility} - \beta \cdot \textit{cost}$$

where β is a parameter that determines the relative importance of utility versus cost. The chromosome encoding used is similar to a standard permutation chromosome using decimal encoding; each chromosome consists of a series of agents' unique numbers that are assigned to corresponding goal points based on position within the chromosome. Thus, each chromosome

consists of a series of decimal encoded agent numbers, the order of which determines the order of assignment to goal points. It is noteworthy that this particular encoding can not represent an assignment of multiple agents to the same goal point. The initial population is randomly generated, selection is fitness-proportional, and one point crossover is used for offspring generation.

Simulation results indicate that this approach produces agent paths that are more coherent and less erratic than standard greedy approaches. Because of the embarrassingly parallel nature of genetic algorithms, it is relatively easy to evenly distribute the large amount of computation involved among participating agents. This approach suffers from similar scalability issues as greedy approaches, however, particularly due to the simultaneous assignment of all agents. In addition, the inability to represent an assignment in which multiple agents pursue the same goal renders this approach incapable of agent coalition formation. This could be rectified by using a different chromosome encoding, and a genetic algorithm approach to goal assignment certainly warrants further study.

2.3.3 Market Based

Significant research has also been done recently in the area of market based allocation strategies. In these strategies, agents negotiate with each other and treat goal points as a commodity that they exchange. Such exchanges are determined by auction mechanisms, though the specific auction mechanism used varies between implementations. In some implementations, single-round single-item sealed-bid auctions that closely resemble greedy allocation strategies are preferred [19]. In other more distinctly market based implementations, multi-round auctions may be used so that bids can account for the effects of previous allocations [7, 9], particularly in the calculation of goal point utility. Occasionally, combinatorial bids are used, in which agents bid on multiple goals in batches rather than individually [8]. The reasoning behind this is that due to spatial proximity of goal points to one another, it is likely advantageous for an agent to pursue groups of goals rather than treat each goal independently.

In most auction based implementations, agents keep a collection of goals, and in many cases maintain a path connecting those goals. This path can be calculated using a TSP heuristic algorithm. A greedy insertion heuristic is particularly well suited due to the frequent insertion of new goals. An example of one such implementation is that described in [19], which uses a variety of mechanisms for goal point selection, none of which are frontier

based. This incorporation of fore-planning into an exploration strategy is intended to provide better individual agent performance and more effective goal point distribution among agents than is provided by purely reactive approaches.

Multi-round Auctions

Multi-round auctions are sometimes used so that agents can incorporate information about previous goal assignments into their bids. This is important for utility calculation (See section 2.3.4). An approach proposed in [7] implements multi-round auctions as a method of distributing a TSP heuristic algorithm. The algorithm presented generates a complete graph of all goal points and agent positions, with edges representing estimated cost to travel between the points they connect. A minimal spanning forest is then generated containing n trees, where n is the number of agents participating. A spanning forest of a graph G is a subgraph of G containing only trees and containing all vertices of G. A minimal spanning forest is a spanning forest that is comprised of a minimum-cost collection of trees. This minimal spanning forest is calculated by iteratively adding vertices to the forest after initializing it to contain only the vertices representing agents' positions. In order to do this, the auctioneer auctions all unassigned goal points simultaneously. Each agent estimates the minimum cost to each of the goals being auctioned by calculating the minimum cost from either the agent's current position, or any other goal already owned by the agent. This approach to calculating cost is unlike many other implementations that calculate all costs based only on an agent's current location. This cost is the value that the agent bids for that particular target. Each agent only submits its minimum bid to the auctioneer, and the auctioneer then allocates only one goal to the bidder with the lowest bid among all agents and goals. Once an allocation is made, the auction is repeated with the remaining unallocated goals. By construction, it can be proved that the resulting collection of tours has a cost of at most twice the optimal solution [7]. It is important to note, however, that even an optimal solution to the Travelling Salesman Problem does not necessarily provide an optimal exploration strategy due to the unpredictable nature of unexplored regions of the map.

Combinatorial Auctions

In combinatorial auctions, agents bid on bundles of goal points rather than single goal points. This approach is motivated by the notion that single

item auctions fail to adequately account for synergies between goal points [2]. Since for a set of n goal points, there are $2^n - 1$ possible bundles on which to bid (a bid on an empty bundle would be meaningless), calculating bids for all possible bundles quickly becomes intractable as the number of goal points increases. In [2], a few possible methods are presented for choosing a limited number of bundles to be considered. One suggested method is to have agents bid on all bundles with no more than three goal points. This approach is very rudimentary and did not perform particularly well in experiments. An alternative method is to use a heuristic similar to a TSP greedy insertion heuristic to build bundles containing goals that are theoretically well suited for successive visiting. This approach performed well in experiments and performed only slightly worse than the graph-cut method. The graph-cut method begins by creating a complete undirected graph whose vertices correspond to the goals to be auctioned and whose edge costs represent the optimistic travel cost between the respective goals. This graph is then split based on its max-cut. This process is repeated recursively on the resulting pair of graphs as long as the graphs contain more than 1 vertex. Since calculating the max-cut of a graph is NP-complete, an approximation algorithm must be used. This graph-cut method performed the best in experiments, but is clearly more computationally intensive to implement. While combinatorial auctions have shown improved performance over single-item auctions [2], they also incur fairly significant computation overhead.

2.3.4 Utility

The prevailing common aspect of various task allocation strategies is the necessity for a way to measure the *value* of a given allocation. In almost all cases, this *value* is calculated by estimating the *cost* and *utility* of the allocation in question. *Utility* is calculated in a variety of ways, most of which estimate the expected area to be revealed by visiting certain goal points. The *value* of an allocation is then given by

$$value = utility - \beta \cdot cost$$

where β represents a coefficient representing the relative values of *cost* and *utility*, since they are generally represented in incompatible units. The *value* of an allocation may be used to choose an allocation greedily, or by agents to make bids in a market based allocation scheme. Below are some examples of various *utility* mechanisms that have been implemented.

Quad-tree Leaf Size

While most recent robot exploration algorithms have focused on frontiers, this is not the only way to generate goal points for exploration purposes. One alternative approach is to perform a quad-tree breakdown of the unexplored regions of the map, and then to use the centers of the leaf nodes of the resulting quad-tree as goal points [19]. A quad-tree is a tree structure used to represent an environment in a variable resolution manner. In a quad-tree, the environment is partitioned by recursively subdividing regions into smaller regions as necessitated by the presence of obstacles. A convenient measure of *utility* for each goal point is the area of the quad-tree leaf node to which it corresponds. This type of approach requires more attention to dynamic path-planning and *cost* updates as more area is explored due to the nature of planning paths through unexplored regions. A similar mechanism may be used for frontier-based exploration, in which the size of the frontier may be used as a measure of *utility*, but this is an extremely rudimentary measure and is generally disregarded in favor of better ones.

Expected Visibility Range

Expected Visibility Range is a measure of the openness of an environment, and is used to adjust the utility of goal points based on the assignment of agents to other nearby goal points. This mechanism is intended to provide a measure of utility that adapts to the nature of the environment being explored, but does not provide different utilities for any goal points that are not near other goal points. Since most hardware platforms used for robotic exploration and mapping use range-based sensors, such as sonar or laser, it is easy to count the number of times a given distance to an obstacle is measured. Using this information, it is possible to compute the probability that a cell will be visible from a goal point given the distance between the two [3]. This probability can be calculated by using the formula

$$P(d) = \frac{\sum_{d_i \leq d} h(d_i)}{\sum_{d_i} h(d_i)}$$

where $h(d_i)$ is the number of times the distance d_i was measured by any of the agents, for a discrete set of distances $d_1, d_2, \ldots, d_n$. The utility of all goal points is then initialized to 1. Upon each successive assignment of some goal $\langle x, y \rangle$ to an agent, the utility of each remaining unassigned goal $\langle x', y' \rangle$ is updated according to the formula

$$U_{x',y'} \leftarrow U_{x',y'} \cdot (1 - P(\| \langle x, y \rangle - \langle x', y' \rangle \|))$$

Expected Information Gain

When frontiers are used as goal points, it is guaranteed that there is some unknown space adjacent to each goal point. By flood-filling over this unknown space and limiting the flood-fill to the visibility range of a given agent, an estimate can be obtained of how much unexplored space would be revealed by that agent were it to be at that particular goal point, resulting in the *expected information gain* of that goal point. A bounding rectangle can then be used as an approximation of that area, and is referred to as the *information gain region*, or IGR [13]. The utility can then be discounted according to the overlap of these rectangular regions, using the formulas

$$d_j = \frac{Area\left(IGR_j \cap \left(\cup_{i \in R} IGR_i\right)\right)}{Area(IGR_j)}$$

$$u_j = (1 - d_j) \cdot i_j$$

where u_j is the utility of goal point j, d_j is the percentage overlap of IGR_j and the rectangular *information gain regions* for the goal points previously assigned to agents R, and i_j is the *expected information gain* of goal point j. In experiments, the use of rectangular regions to approximate overlap produced results within 15 percent of the true overlap, obtained by counting the actual number of overlapping cells.

2.4 Aspects Utilized in Experiments

In this chapter, many aspects of robotic exploration have been discussed. Not all of those aspects were directly involved in the simulation experiments performed. For example, evidence grids normally contain probabilities that a given cell contains an obstacle as a mechanism to accommodate sensor uncertainty, but the evidence grids used in simulation experiments were binary. In order to simplify the simulation platform and avoid introducing potential factors that might skew results, sensor uncertainty was not simulated. Instead, agents were given a visibility range, and could see any obstacles within that range and in any direction with perfect accuracy. Evidence grids are still used, however, as they provide a convenient map representation that is likely to be available in real exploration implementations, and enable the use of frontiers as a goal point generation mechanism. Similarly, many of the more complex market based strategies presented were not directly used by the exploration algorithms in experiments, but rather were influential in the formulation of the hierarchical task allocation algorithm presented in

Chapter 4. Genetic algorithms were not used at all in the development of the hierarchical allocation algorithm, but rather serve as a demonstration of the variety of exploration strategies that have been explored.

Chapter 3

Topology

In order to hierarchically distribute the goal points to be visited among the agents participating in the exploration, the topology of the known environment is used to constrain agent participation in auctions. Initially, it was considered that the topology might be used directly to hierarchically allocate goal points to agents, but this was abandoned in favor of the market architecture discussed in Chapter 4. Maintaining a topological graph of the environment proved to still be useful, however, as it could be used to measure of how far an auctioneer is from potential bidders, so that agents that are sufficiently distant from the auctioneer are prevented from even participating in the auction. This distance threshold can be incrementally increased until a bid is received that renders any further increase not likely to produce a better bid. Since bids are indirectly based on the distance between auctioneer and bidder, a topological graph that incorporates distance is expected to be a reasonable heuristic for pruning the participation of agents in auctions. Such a topology can be calculated in a variety of ways.

3.1 Voronoi

One method of generating a topological graph of an environment is via Generalized Voronoi Diagram (GVD). In mathematics, a Voronoi diagram, or Voronoi decomposition, is a decomposition of a metric space based on distance to a specified set of points in that space. For a set of points S in Euclidean space, every point in space is either closer to one point in S than all others, or is equidistant to two or more points in S. Points that fall into the latter category form edges and vertices of the Voronoi diagram, so that points equidistant to exactly two points in S form edges, and points

equidistant to three or more points in S form vertices. Fortune's algorithm is a plane sweep algorithm that calculates a Voronoi diagram in $O(n \cdot log(n))$ time [4].

In robotics, however, the GVD is used, which varies from the traditional Voronoi diagram in that the set of points S above is no longer a set of points, but rather of obstacles. These obstacles are generally polygons, though they are not necessarily stored internally as such. Because of this generalization, it is difficult to apply Fortune's algorithm to evidence grid maps. Each obstacle cell can be treated as a single point, but this produces a large number of extra edges between points that must then be pruned. Fortunately, the evidence grid map representation model lends itself to another simple way of calculating the GVD. That is, by flood-filling over open space in the environment, the nearest obstacle can be calculated for each cell, and cells that are approximately equidistant to two or more obstacles can be readily identified. In addition, this approach lends itself conveniently to an incremental implementation that is specifically useful for exploration. For an example of a GVD generated by an evidence grid map, see Figure 3.1.

In the case of exploration, when the environment is incrementally revealed, the GVD must be calculated incrementally to account for new information. While calculating the GVD incrementally requires additional space overhead, the time saved by avoiding recalculation of the entire GVD outweighs the cost. This incremental calculation can be accomplished by treating frontiers as obstacles, and marking cells for recalculation whose nearest obstacle is a frontier. The need to retain such a set of cells marked for recalculation accounts for the additional space required for incremental calculation. This process produces a cell-based representation similar to the evidence grid map representation. The Generalized Voronoi Graph (GVG) can be extracted from this representation by a similar flood-fill algorithm that iterates over only the cells comprising the GVD. The GVG is simply the graph representation of the GVD, represented by vertices and edges instead of grid cells. The GVG therefore loses information regarding the shape of paths that connect vertices, but retains the lengths of those paths as weights associated with edges in the graph.

3.1.1 Flood-fill

For each cell in the evidence grid, a 5-tuple is maintained, (x, y, δ, u, v), where (x, y) is the location of the cell, δ is the distance to the nearest obstacle, and (u, v) is the location of the obstacle to which the cell is closest, and is referred to as a *contact* of the cell. A priority queue, Q, of cells ordered

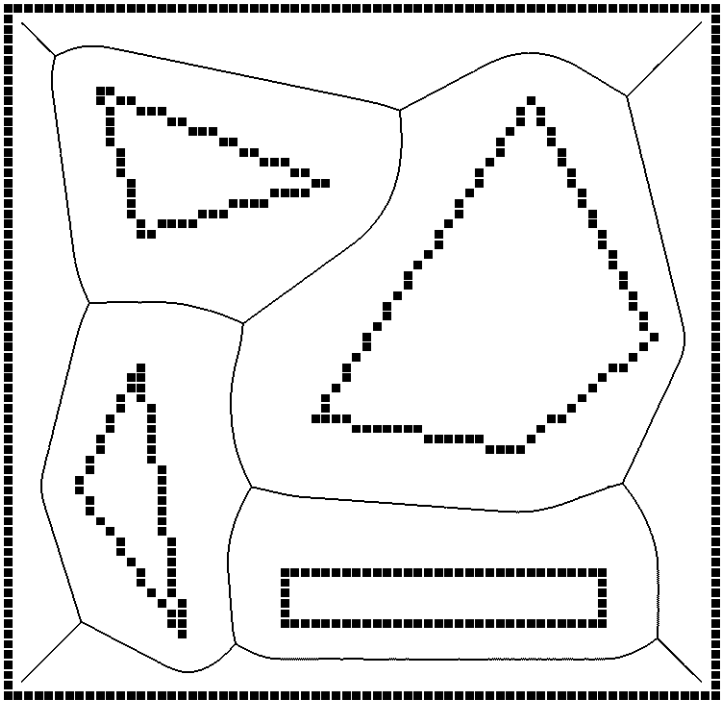

Figure 3.1: Generalized Voronoi Diagram (GVD) generated from an evidence grid map

on δ is then initially populated with obstacle cells, which have $\delta = 0$, and $(u, v) = (x, y)$. Each cell in Q then updates its neighbors, and enqueues in Q any neighbors whose values for δ, u, or v changed. When a cell A updates a neighbor cell B, the current value of δ for B is compared with B's distance to A's nearest obstacle (u, v). If A's nearest obstacle (u, v) is closer to B than B's current value of δ, then B's values for δ, u, and v are updated to reflect this. In addition, if A would cause an update to B, but A's nearest obstacle (u, v) is sufficiently far from B's nearest obstacle (u, v), then A and B are considered to be on the GVD, and B is not added to Q for further updates.

Because frontiers represent unknown space, they are treated as obstacles for the purposes of calculating the GVD. Any cells for which (u, v) is flagged as a frontier are not persisted between updates, so that those cells can be updated with new information in subsequent updates. In order to correctly

expand the GVD when new information is known, persisted cells that border those unpersisted cells must be saved. When the next update occurs, those cells will be added to Q during initialization. In addition, obstacle cells that are adjacent to unexplored cells must be saved and added to Q during the following initialization phase in a similar manner. If this is not done, then newly revealed open cells adjacent to those obstacles will be incorrectly processed because the obstacles will be considered old territory and will not be enqueued. Sections of the GVD for which one or more contact points are frontiers are disregarded when the next update occurs. See Figure 3.2 for an example of this. In the figure, the hatched regions indicate explored cells that are closer to a frontier than any other obstacle. These regions and the Voronoi paths adjacent to them will be cleared when new area is explored. All other open space has been completely processed and will not be recalculated again. This approach would need to be modified to accommodate dynamic environments.

3.1.2 Graph Representation

In order to extract topological information from the GVD, it must be represented as a graph. The flood-fill method above produces a grid of cells that comprise the GVD, and is able to differentiate vertices from edges by keeping track of how many neighbors update a cell with a contact that is sufficiently distant from all current contacts. This entails associating a set of contacts with each cell that is on the GVD, in addition to the 5-tuple described above. That is, for each cell on the GVD, a set is maintained of the (u, v) values of the cells that have updated it. When a new cell updates a cell on the GVD, the new cell's (u, v) values are compared against those currently in the set. If the new cell's (u, v) values are sufficiently far away from all current set entries, then the new (u, v) values are added to the set. In this way, the number of elements in this set is used to determine the location and degree of vertices in the graph representation of the GVD. A simple flood-fill over the remaining non-vertex cells in the GVD is used to determine connectivity between vertices, thus producing the Generalized Voronoi Graph (GVG) from the GVD.

3.1.3 Analysis

The Voronoi approach to topology representation produces a very minimal graph, with vertices only where the environment splits. This is advantageous as it reduces the computation time required when calculating distances be-

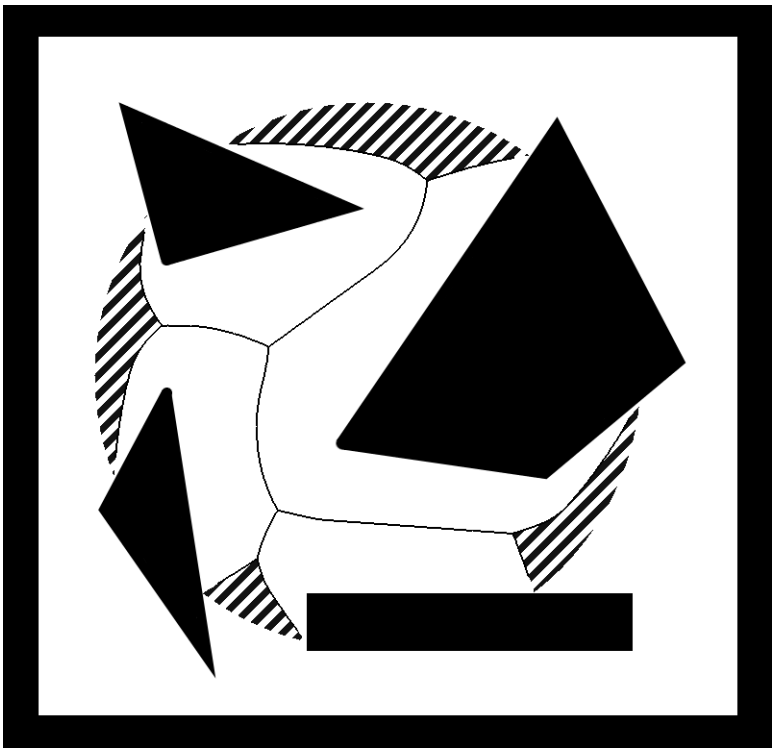

Figure 3.2: An example of frontier cells in the GVD calculation that must be reprocessed when more map data is available.

tween nodes in the topological graph. While it is guaranteed, however, that every point in the environment can see a point on the GVD, it is not guaranteed that every point in the environment can see a vertex on the GVD (See Figure 3.3 for an example of such a case). This can be solved by keeping track of the edge, if any, to which each cell belongs, including the vertices that the edge connects. By doing this, points that can only see an edge on the GVD can still be connected to the graph representation thereof.

3.2 Probabilistic Road-map

Probabilistic road-maps (PRMs) have been successfully implemented for path planning [6], and their computation has been shown to be embarrassingly parallel [1]. This makes them especially convenient for cooperative robotic exploration, since as much of the computation as possible should

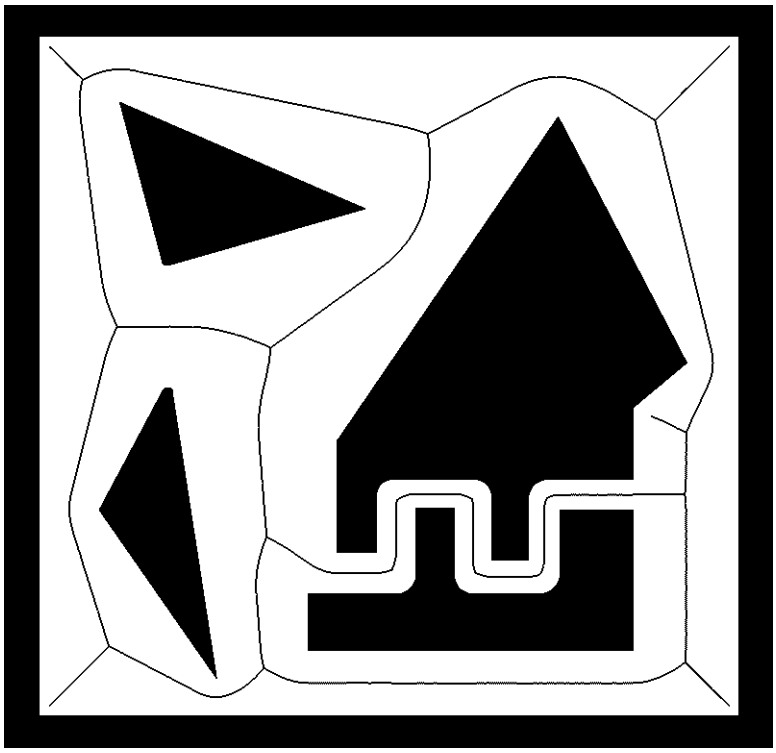

Figure 3.3: An example of a scenario in which points in the environment do not have a line-of-sight to a vertex in the GVD. Specifically, there are such points in the middle of the path through the largest obstacle shown.

be distributed to take advantage of all available resources. Traditionally, a PRM is built in the free configuration space of a robot by generating random points and interconnecting them according to some restrictions that vary between implementations. Start and end points are then added to the PRM in the same way as each of the randomly generated points. One of the complications that arises in the computation of a PRM is deciding how many random points to generate.

While the PRM approach is traditionally used for path planning, a similar approach can be used to create a graph representation of the topology of an environment. This type of topological representation is particularly simple to update as exploration progresses, as this can be done simply by generating new points in the newly discovered areas. A PRM topology representation is significantly more dense than a Voronoi-based representation.

This increases the computation time when using the topology graph, but also makes it easier to add agents' positions to the graph. This type of approach was considered as an alternative to the more complex Voronoi method discussed above, but was not used in favor of the incremental Voronoi calculation method.

Chapter 4

Hierarchical Task Allocation

This chapter provides a discussion of the task allocation methods employed to perform hierarchical task allocation for exploration purposes. A market based mechanism is used, the details of which are discussed further below. Each agent takes on the role of either a *supervisor* or a *worker*, the details of which are described in section 4.2. While an agent is deciding what to do next, it is in a third state, *retasking*. An agent is always in exactly one of these three states. See figure 4.1 for a representation of the transitions between these states.

There are two independent auction mechanisms that take place to perform task allocation. The first is a *Goal Auction*, in which agents auction and bid on goals. The second is an *Agent Auction*, in which an agent auctions its services in the event that it does not have its own goals to pursue. These two auction mechanisms take place asynchronously, but care must be taken so that an agent does not transfer a goal in a *Goal Auction* that it has used to make a bid in an *Agent Auction*. If this were to happen, a second agent may accept the bid in the *Agent Auction* and join the first agent, despite the fact that the first agent would be no longer responsible for that goal. These auctions continue taking place as long as their are goals available. Depending on what goal generation technique is being used, this may result in exploration of all reachable open space, or only specific regions. In experiments performed, all reachable open space was explored.

4.1 Goal Auctions

In order for an exploration strategy to be truly distributed, there must be a sharing of responsibility for goals among agents. The agent initially respon-

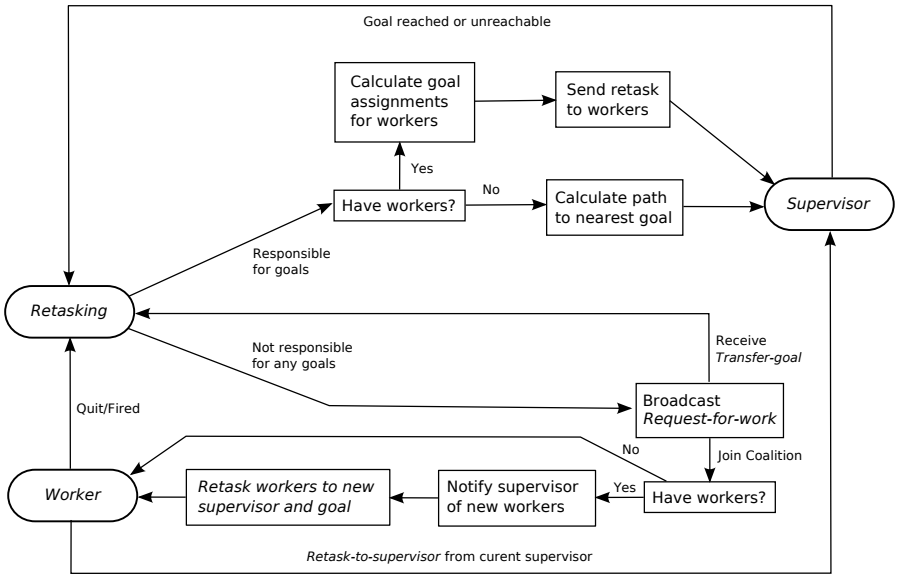

Figure 4.1: State diagram describing the transitions between *supervisor*, *worker*, and *retasking* states.

sible for a goal is trivially the agent that discovered the open space to which the frontier is adjacent. Frontiers were used for generating goal points in all experiments performed, but this goal auction method is intended to function equally well for other methods of goal point generation. For non-frontier goal points, similar mechanisms can be chosen so that agents generate their own respective goals. During the course of exploration, however, the agent that generated a goal point may become no longer the most optimal agent for exploring that goal point. In such cases, a mechanism for agents to transfer goal points between them increases exploration efficiency.

A market architecture provides a simple and effective way for agents to transfer responsibility for goal points. When an agent generates new goal points, usually by reaching a current goal point, it can hold an auction so that goal points may be transferred to more optimal agents. An agent may also periodically hold auctions even when no new goals are discovered so that goals can still be transferred between agents during long paths through explored regions. While many complex auction strategies exist (See section 2.3.3), simple single-round highest-bidder closed auctions can be used for

distributing responsibility for goals among agents.

4.2 Agent Auctions

Definition 4.1. A *coalition* is a set of agents simultaneously moving to explore the same goal point and is comprised of exactly one *supervisor* and zero or more *workers*.

In order to hierarchically distribute goals to agents, agents form coalitions. When an agent is not responsible for any goals itself, it can obtain a goal from another agent. In some cases, this entails joining another agent in a coalition. Coalitions necessarily form when there are more agents than available goals, which is often the case in highly structured environments. When a coalition has been formed to pursue a goal and the resulting exploration of that goal reveals two or more new goals, that coalition will divide into smaller coalitions so that the newly generated goals can be effectively explored. In this way, coalitions hierarchically divide and allocate tasks accordingly.

Definition 4.2. The *supervisor* of a coalition is the agent responsible for that coalition's goal. Thus, every coalition has exactly one *supervisor*.

Definition 4.3. *Workers* are non-supervisor agents belonging to a coalition. A coalition may have zero or more *workers*.

Workers are agents who have joined the supervisor because they do not have any goals of their own to pursue. Each non-worker agent is therefore trivially the supervisor of a coalition of size one – the coalition containing only that agent. Agents that are workers do not have any goals for which they are responsible and therefore do not hold auctions to transfer goals to other agents and cannot be supervisors. In addition, an agent may only belong to exactly one coalition at any time, though it may be either the supervisor or a worker in that coalition. Agents that are supervisors must be responsible for at least one goal. The supervisor of a coalition is responsible for notifying the workers of that coalition of any changes to the current goal of the coalition. That is, when the goal of a coalition is explored, a new goal must be determined and the workers must be informed of this change.

4.2.1 Coalition Formation

Coalitions of more than a single agent are formed by agents that do not have their own goals to pursue and elect to assist other agents. In particular, such coalitions will necessarily be formed when there are more agents than goals. The mechanism used to do this is similar to the market used to allocate goals to agents. Instead of holding an auction to transfer goals to bidders, however, an agent holds an auction seeking the most profitable goal for it to pursue by broadcasting a *Request-for-work* message. In this way, an agent may obtain goals that it would not otherwise. Alternatively, if it is more profitable or necessary because of a shortage of goals, the agent holding the auction may join an existing coalition. See Figure 4.2 for more details regarding the interaction between an auctioneer and bidders that takes place in response to a *Request-for-work* message.

In order to compare joining a coalition to alternative courses of action, an expected profit must be calculated for a potential coalition. The profit of a coalition can be calculated by

$$Profit = \frac{Max_{i \in A}(Utility(i,g)) - \beta \cdot \sum_{i \in A} Cost(i,g)}{\|A\|}$$

where A is the set of agents belonging to the coalition and g is the coalition's goal. In the case where this results in a negative value for *Profit*, the formula

$$Profit = \left(Max_{i \in A}(Utility(i,g)) - \beta \cdot \sum_{i \in A} Cost(i,g) \right) \cdot \|A\|$$

must be used so that larger coalition sizes are penalized rather than rewarded. Since there is no way to know how many avenues of exploration a goal will produce (See Section 6.4), it is assumed that smaller coalitions provide a more even distribution of workers among available goal points and are therefore desirable.

4.2.2 Coalition Maintenance

In many cases, particularly in highly structured environments, exploration of a goal point produces only a single new goal point. In these cases, it is sensible for a coalition to continue on to the new goal. This is accomplished by the supervisor sending a retask message to coalition workers informing them of the new goal to pursue. Each worker then calculates a path to that goal. A worker may leave a coalition at any time, but this retasking provides a particularly opportune time for workers to consider alternative

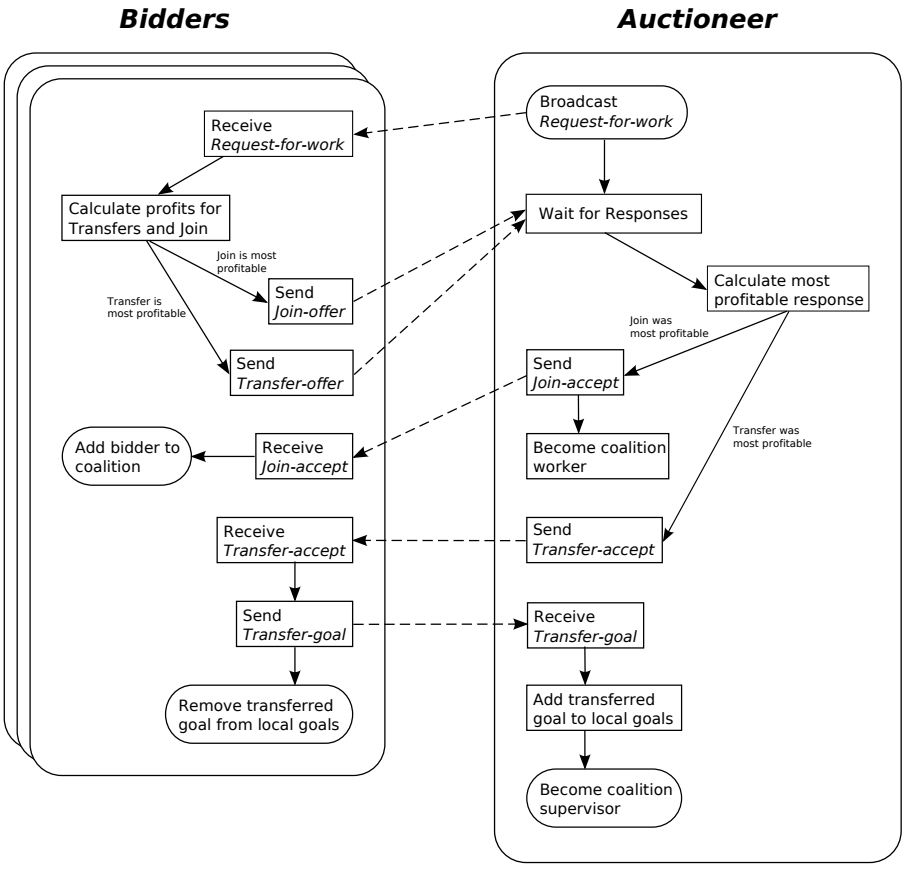

Figure 4.2: *Request-for-work* mechanism, initiated by an agent that does not have any of its own goals to pursue.

courses of action. A worker can quit a coalition at any time by informing the supervisor of its intent to quit. It is important, however, that all workers in a coalition do not simultaneously quit the coalition to broadcast work requests. The resulting large number of simultaneous work request broadcasts would hinder the effectiveness of the greedy market based allocation process.

The case in which a coalition explores a goal that results in multiple new goals requires particular attention. If a worker discovers a new goal point, that worker will quit the coalition and become its own supervisor. It will then respond to future *Request-for-work* broadcasts to obtain its own workers. If the supervisor of a coalition discovers multiple goal points, it is the supervisor's responsibility to decide which workers will pursue which goals. This can be done greedily in the following manner. First, agents are ordered by topological distance to the supervisor, including the supervisor, which trivially has a distance of zero. Each agent is then greedily assigned to the most profitable goal for that agent, beginning with the supervisor. In this way, the supervisor will pursue its most profitable goal, and other agents will be assigned to other goals if such an assignment is more profitable than joining the supervisor. The first agent assigned to a goal will become a supervisor responsible for that goal, and any further agents assigned to the same goal will be transferred to the new supervisor as workers. Note that agents assigned to be supervisors in this manner may refuse to take part in a coalition by firing all workers, but may not refuse responsibility for the new goal point. Once an agent has become its own supervisor, it is no longer affiliated with the agent that was previously its supervisor.

In order to accommodate all these interactions, three types of retask messages are required. They are *Retask-simple*, *Retask-become-supervisor*, and *Retask-change-supervisor*. A *Retask-simple* message simply instructs a worker to calculate a path to and pursue a new goal point. A *Retask-become-supervisor* message instructs a worker to become a supervisor that is responsible for the included goal point. Implicitly, the new supervisor is to calculate a path to and pursue the new goal. A *Retask-change-supervisor* message instructs an agent to join a new supervisor's coalition. Upon doing so, the worker will be given a new goal point to pursue.

4.2.3 Coalition Dissolution

Coalitions may be dissolved for a number of reasons. In very open environments, it is not likely that coalitions will even form once the agents have dispersed sufficiently. A worker may choose to quit its current coalition and reevaluate a new task at any time. This may result in rejoining the same

coalition the agent just quit. This is particularly useful when the worker is far away from the coalition's goal, since the state of the exploration changes over time, and it is possible, if not likely, that a better alternative will arise for the worker. A worker may also receive its own goal, either by revealing newly explored open territory, or by bidding in goal auctions (Workers do not hold goal auctions because they have no goals of their own to auction, but they still bid in goal auctions held by other agents). In such a case, the worker will quit the coalition and pursue its own goal.

It is also possible for the supervisor to dissolve a coalition. In cases where the exploration of the coalition goal results in more than a single new goal point, the supervisor will decide which agents will receive which goals and which coalition each agent will join. In this case, the previous coalition will decrease in size, potentially even to become a trivial coalition of just the supervisor. Alternatively, the exploration of the coalition goal may result in no new goals. If a situation arises in which the supervisor is no longer responsible for any goals, it will make use of the *Request-for-work* mechanism (See Figure 4.2) to obtain a task. If this results in the supervisor joining another coalition, it will become a worker itself and will transfer the workers of the old coalition to its new supervisor by sending them *Retask-change-supervisor* messages. It is also necessary to notify the new supervisor of the addition and the workers of the change in supervisor, by sending it a *Transfer-workers* message.

Chapter 5

Experiments

This chapter provides a description of the experiments used to test the performance of the previously described hierarchical task allocation method and the results thereof. Experiments were performed on a variety of maps with different numbers of participating agents. A rudimentary greedy algorithm was implemented as a control group for comparison purposes, and is described in further detail below. The results of the experiments are presented and discussed in this chapter, as well as conclusions drawn from those results.

5.1 Simulator

In order to conduct exploration experiments, a simulator has been written in Java that communicates with clients via plain-text messages over TCP/IP. The simulator notifies clients of newly explored area and provides messaging between clients in both point-to-point and broadcast manners. The server notifies clients of all newly explored territory, regardless of which client explored the territory. While this sort of centralized approach may seem to contradict the distributed nature of the exploration algorithms being tested, distributed map storage is beyond the scope of this thesis and would not effect the exploration algorithms being tested any differently. Communication between clients and the server is asynchronous. The server sends regular updates to clients, and clients may send messages to the server at any time. All messages between clients are sent through the server. Clients are not provided with any means of contacting other clients directly.

The simulator uses the flood-fill algorithm discussed in Section 3.1.1 to calculate the topology of the known environment according to the GVD of

that environment. This topology is then used to optionally limit the range of broadcast communications. Initially, the topology was intended to be used for coalition formation purposes, but was disregarded in favor of a market based approach. While it is not currently used for this purpose, the topology could also be employed for faster path-planning and topological distance calculations. In addition, the topology may be used to predict the probability that a particular avenue of exploration branches (See Section 6.4).

5.2 Maps

Experiments were performed on a set of four maps. The maps were created to represent a variety of environments. The maps vary in obstacle density and structure, but are all the same size. The maps have a total area (obstacles and open space) of approximately 100 times the visible area of a single agent. That is, maps are represented as 800 by 600 pixel bitmaps, and agents have a radius of vision of 40 pixels. The first map is devoid of obstacles, except for a boundary preventing agents from reaching the edge of the map. The second and third maps contain increasingly many randomly sized, shaped, and positioned obstacles. The fourth map is a highly structured map specifically designed to test the performance of exploration methods in an inherently hierarchical environment. Both the greedy test algorithm and the hierarchical experimental algorithm were tested with varying number of agents in each of the four maps. The numbers of agents tested were 1, 2, 4, 8, 16, and 32. See Figure 5.1 for the four maps used.

5.3 Algorithms

Experiments were performed using both the algorithm described in 4 and a rudimentary greedy algorithm. This greedy algorithm was implemented as a control group in order to demonstrate the feasibility of hierarchical task allocation. In this rudimentary greedy algorithm, each agent is responsible for the goals to which it is closer than any other agent. An agent pursues whatever goal it has that is the closest topologically. If an agent is not responsible for any goals, it will broadcast a request to other agents and will pursue the goal it receives that is the closest topologically. Note that this mechanism does not transfer responsibility for the goal, it merely provides the agent with an interim goal to pursue until it is responsible for its own goal rather than remaining stationary.

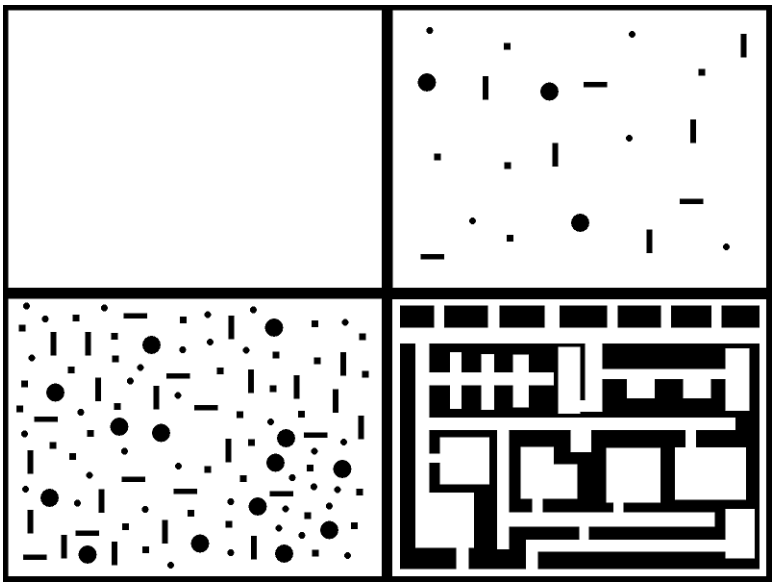

Figure 5.1: The four maps used for experiments: open, sparse, dense, structured

This rudimentary algorithm is very simple, entailing minimal cooperation between agents. These experiments are intended to provide a demonstration of the potential benefits of incorporating coalition forming techniques into current exploration strategies. They are not intended to provide a comparison of the coalition forming algorithm with other contemporary exploration techniques. In addition, experiments were all performed on homogeneous groups of agents for simplicity, but coalition forming techniques may prove increasingly useful in more heterogeneous systems (See section 6.3). The parameters for both algorithms' experiments were the same. That is, all agents started in the center of the map for all tests, and all agents had the same speed and visibility range. Exploration was completed by revealing all reachable open space in the map. In addition, both algorithms used the same mechanism for identifying goal points. That is, frontiers were used in both cases. Frontiers are identified as connected groups of open cells adjacent to unexplored cells. Goal points are then determined by calculating the arithmetic mean of the points comprising the frontier. In the case of an extremely large frontier, such as necessarily forms in the open environment, the frontier is divided into smaller frontiers that each generate a respective goal point.

5.4 Results

As expected, results varied between map types, particularly between the random maps and the structured map. Hierarchical allocation methods did not perform significantly worse than the greedy test implementation in any of the tests. More improvement was seen on less open maps, with the most improvement seen on the structured map. Multiple measures of performance are used to evaluate exploration strategies, a few of which are addressed further below. In the case of a single agent, the exploration algorithms are effectively identical, so this case will not be presented, though the experiments were performed. This section presents some representative data and graphs. More comprehensive results can be found in Appendix A.

5.4.1 Area Explored Versus Time

This is perhaps the most intuitive measure of the performance of an exploration algorithm. It is presented as a graph of the map area explored versus time. Since the simulator calculates its state in increments, henceforth referred to as ticks, the units of time used in the graph are arbitrary and correspond to simulator ticks rather than any wall time unit. Area explored is simply measured in pixels, since the exploration environment maps are loaded as bitmaps, providing pixels as a convenient measure of area. There are slight discrepancies between the algorithms' maximum area explored in some cases. This is due to the occurrence of unexplored regions that are too small to be considered frontiers and are therefore unexplored when the exploration is considered complete. These regions are not significant in the comparison of the exploration algorithms, as both algorithms used the same method to identify goal points. See Figures 5.2 and 5.3 for Area Explored Versus Time graphs for the dense and structured environments respectively. These figures highlight the difference in performance of the hierarchical allocation method between the random maps and the structured map. The performance among the open, sparse, and dense random maps was less varied. See Figures 5.4 and 5.5 for Area Explored Versus Time graphs of the open and sparse environments.

5.4.2 Time Required for Percentage Completion

This measure of performance is derived from the same data as the Area Explored Versus Time measure, but is presented in a slightly different format graphically. This measure provides a graphical representation of the

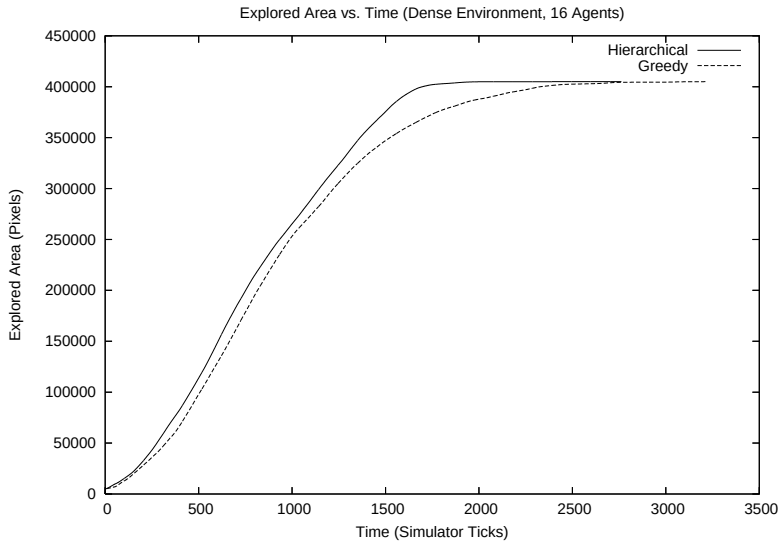

Figure 5.2: Area Explored Versus Time graph for 16 agents exploring the dense environment

amount of time required to explore various percentages of the environment. Again, time is measured in simulator ticks. This measure of performance emphasizes differences in performance at particular stages of exploration. See Figures 5.6 and 5.7 for graphs of Time Required for Percentage Completion for the dense and structured environments respectively.

5.4.3 Qualitative Observations

Qualitative observations do not provide concrete support for the validity of hierarchical task allocation, but they can help explain the quantitative observations made and provide some insight into future ideas worth pursuing. The hierarchical allocation method more effectively allocates agents when the agents start in a large group and must divide to explore new goals. When the agents are already dispersed, however, and there are few remaining goals to explore, the hierarchical allocation method did not perform as well as the greedy approach. It is unclear why this was, but may potentially be due to

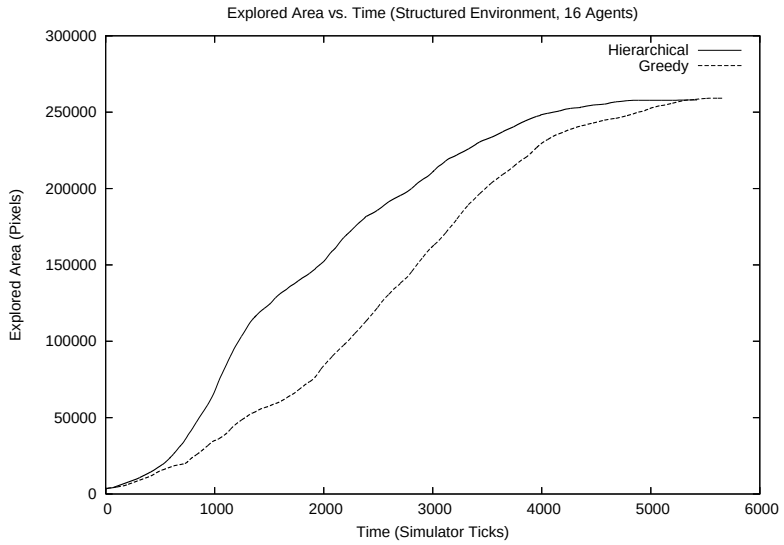

Figure 5.3: Area Explored Versus Time graph for 16 agents exploring the structured environment

computational delays in calculating costs of joining agents far away.

5.5 Discussion

Results indicate that hierarchical coalition forming task allocation techniques for robotic exploration perform better than simple greedy approaches. This is particularly the case in very dense or structured environments, but even in open space, hierarchical task allocation performs no worse than greedy allocation. This chapter provides some conclusions regarding the performance hierarchical task allocation and the various contributing factors thereof. The primary contributing factors that are discussed are map type, number of agents, and exploration stage.

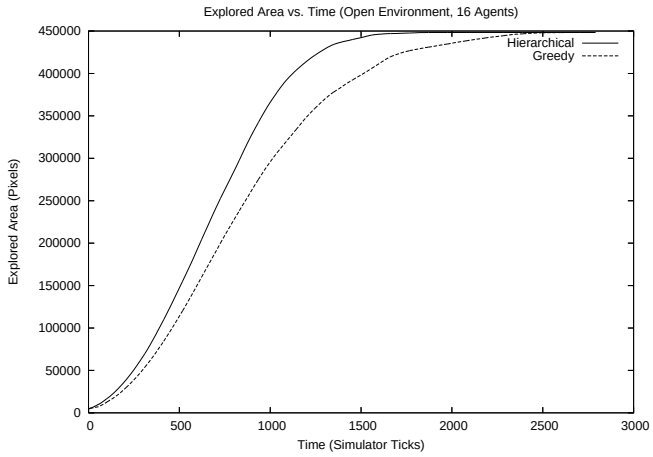

Figure 5.4: Area Explored Versus Time graph for 16 agents exploring the open environment

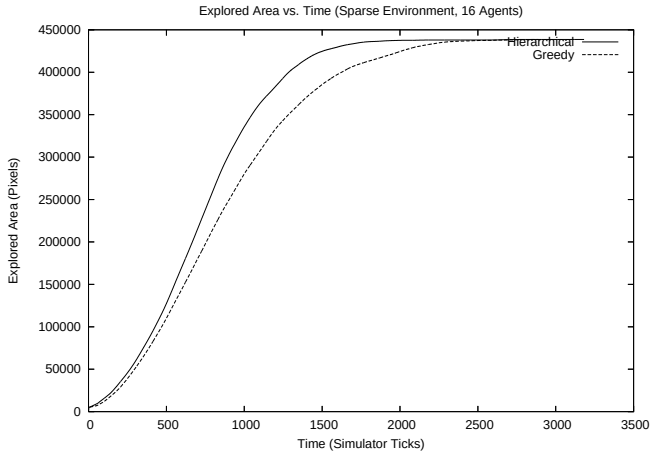

Figure 5.5: Area Explored Versus Time graph for 16 agents exploring the sparse environment

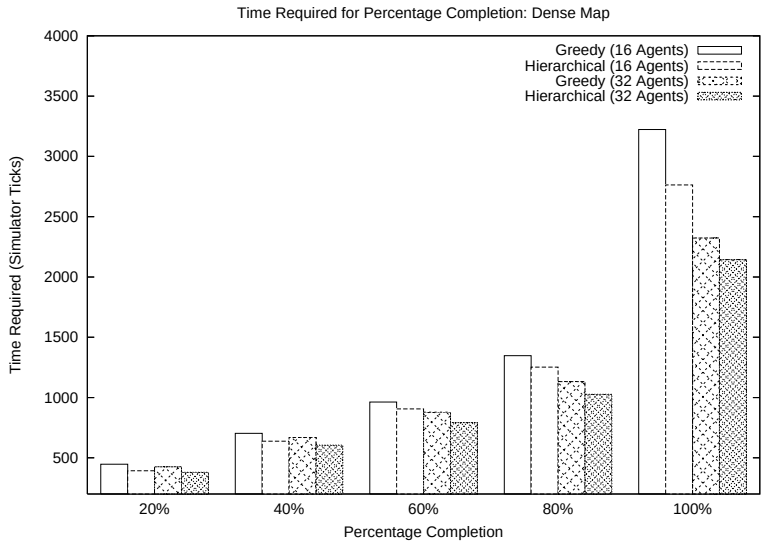

Figure 5.6: Time Required for Percentage Completion for 16 and 32 agents exploring the dense environment

5.5.1 Map Type

Map type significantly affected the performance of the hierarchical task allocation algorithm in the experiments that were performed. In the open environment, the hierarchical allocation performed about as well as the greedy test algorithm, particularly once agents had spread out and were no longer working in coalitions. The greedy test algorithm is only slightly less naïve than completely independent exploration in which agents do not communicate, so this is not a particularly good performance. This reinforces the notion that hierarchical allocation methods alone are insufficient for a generic exploration strategy.

In order demonstrate the difference in performance gain between different map types, area explored and exploration time must be expressed as percentages, since the map types vary in total area and total time required for exploration. See Figure 5.8 for a comparison of the open and structured environments in this manner. Percentage difference in area explored

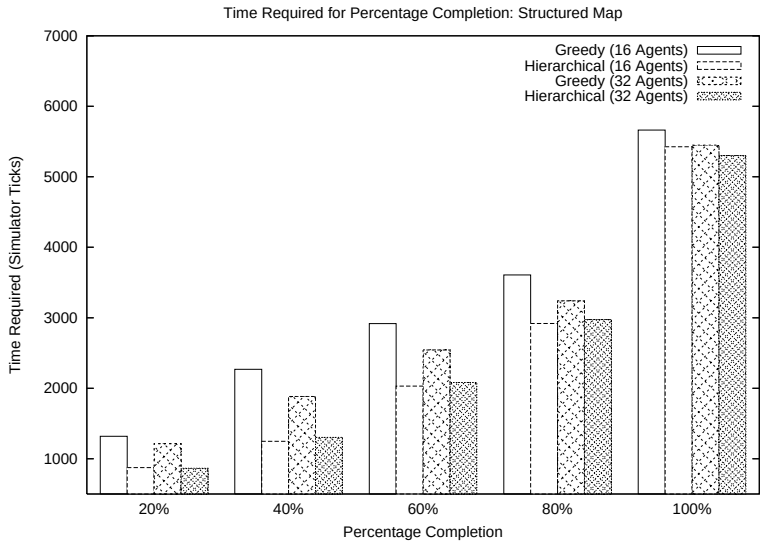

Figure 5.7: Time Required for Percentage Completion for 16 and 32 agents exploring the structured environment

is calculated as

$$P_d = 100 * \left(\frac{A_h - A_g}{A_g} \right)$$

where A_h and A_g are the area explored in pixels of the hierarchical and greedy allocation methods respectively. Percentage of exploration completed is calculated based on the time required for the hierarchical algorithm to complete the exploration. From this graph, it can clearly be seen that the hierarchical allocation algorithm performed much better in the structured environment. In both environments, the hierarchical allocation method performed significantly better in the very early stages of exploration. This is because the hierarchical method allocates agents more effectively when there are more agents than goals. When this is the case, the rudimentary greedy algorithm simply allocates the extra agents by assigning them to their respective nearest goals, rather than distributing them evenly among the goals as the hierarchical algorithm does.

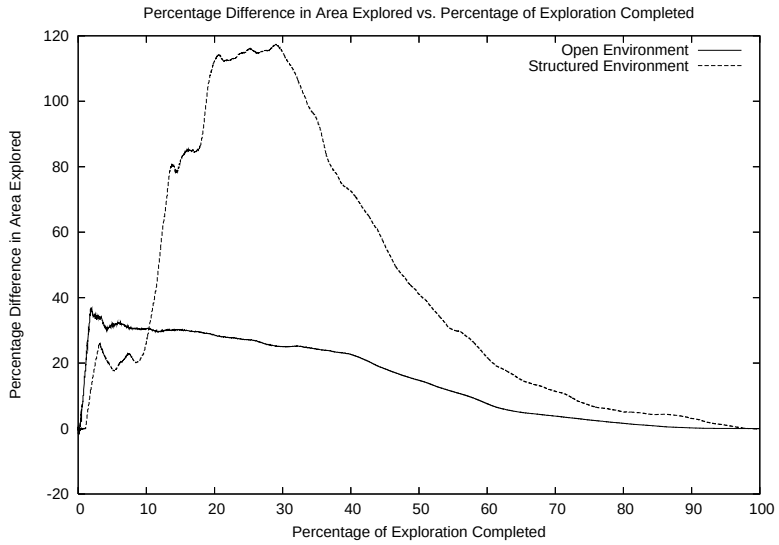

Figure 5.8: Graph of Percentage Difference in Area Explored versus Percentage of Exploration Completed for 16 Agents in the open and structured environments.

5.5.2 Number of Agents

The primary purpose of a multi-agent exploration strategy is to effectively utilize the advantages of having multiple agents explore simultaneously. That is, it is desirable to minimize the amount of backtracking that agents do and the overlap between agents' vision. The performance of an exploration strategy can be measured in terms of the benefit gained by adding additional agents. Ideally, doubling the number of agents participating would halve the time required for exploration. Even with a priori map information, however, it is often not possible to achieve this kind of performance increase. The optimum number of agents for exploring an environment depends primarily on the nature of the environment. The open environment, for example, showed greater improvements in exploration performance when the number of agents was increased from 16 to 32 than the structured environment showed under the same conditions. This can be seen in Figures 5.6 and 5.7.

While the effectiveness of adding agents depends largely on the nature of the map, it also depends on the exploration algorithm. No algorithm will be able to perform better with more agents once the optimal number of agents has been reached, but a good algorithm can more effectively allocate agents when there are fewer than the optimal number of agents available. Figure 5.7 demonstrates this, as the hierarchical allocation method performed about as well with 16 agents as it did with 32 agents. While this could indicate that the hierarchical allocation method did not effectively allocate all 32 agents, it performed better than the greedy algorithm in both cases, so it is more likely that the nature of the environment is such that there is little to be gained by using 32 agents rather than 16, so long as the exploration algorithm is effectively allocating those agents.

5.5.3 Exploration Stages

There are various clearly identifiable stages in the exploration of an environment. One way of distinguishing exploration stages is to identify the time at which the most distant reachable point from the starting position is explored. After this time, agents generally backtrack extensively in order to explore regions that were previously passed by and left unexplored. If an exploration strategy were to operate in a breadth-first manner, then this point would be very near the end of the exploration. In contrast, if an exploration were to operate in a depth-first manner, this point would be nearer the beginning of the exploration. Since neither the hierarchical allocation method being tested nor the greedy test algorithm explicitly operate in a breadth-first or depth-first way, the first stage of exploration ends at a point that depends primarily on the structure of the environment. That is, in a highly structured environment, it is likely that the first stage will take longer since agents must navigate a network of passageways rather than being able to disperse in all directions. In Figure 5.8, it is evident that the hierarchical allocation method performs better in the first stage of exploration. It is unclear why this is the case, but it may be due to the long distances between agents resulting in longer computation times in forming coalitions in the later stage.

Another notable point in the exploration process that applies primarily to the hierarchical allocation method is the point at which there are more goals to be explored than there are agents. While it is possible for coalitions to form even when there are more goals than agents, it is significantly less likely, depending largely on the constant (β) used in the utility calculations (See Section 2.3.4). Once agents are no longer forming coalitions, the hier-

archical allocation method essentially becomes the same as the greedy test algorithm. In some environments, this may never occur. For example, in the structured environment and with 16 agents exploring, there were always coalitions of more than one agent. This can be seen in Figure 5.9. In this figure, Area Explored and Coalitions Formed are represented as percentages. Coalitions Formed is calculated as the number of coalitions that exist at any given time divided by the number of agents, since the number of agents is the maximum number of coalitions that may form. This figure also demonstrates the decrease in rate of area being explored as the number of coalitions drops significantly around 70% through the exploration. In instances when there are more goals than agents, there will eventually be fewer goals than agents again, usually near the end of the exploration. The hierarchical allocation method does not appear to perform particularly well once there are more goals than agents, even after the number of goals decreases back below the number of agents. This can be seen in Figure 5.8.

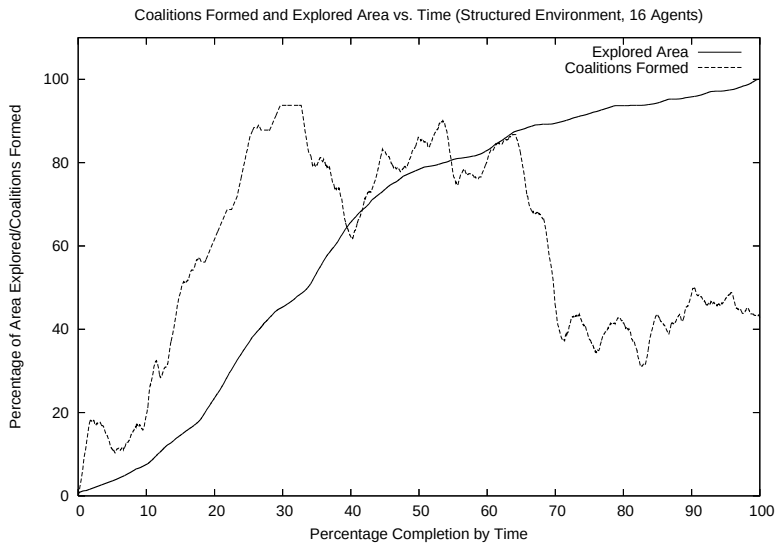

Figure 5.9: Area Explored and Coalitions Formed Versus Time graph for 16 agents exploring the dense environment

5.6 Conclusions

In conclusion, hierarchical task allocation has shown potential to provide improvements to existing exploration techniques. Hierarchical allocation alone, however, appears to be insufficient as a generic exploration strategy, as the benefit it provides varies greatly with the nature of the exploration environment. Some potential ways to incorporate hierarchical allocation with other modern exploration strategies are discussed in the next chapter.

Chapter 6

Future Work

This chapter provides some brief descriptions of various future directions in which research in hierarchical task allocation may be taken. This sort of allocation mechanism could be combined with many existing allocation techniques. In addition, a more domain-specific implementation, such as one using a more involved coalition utility functions, may provide more impressive results.

6.1 Tour Planning

The algorithm presented determines a goal for each agent to pursue, but it does not plan ahead for future goals. Certain exploration algorithms [19] maintain a tour of goals for each agent, in an attempt to incorporate a notion of foresight into the algorithm. While this type of approach may be less effective for coalitions than individual agents due to the potential for coalition division, it may still provide some improvement over the more naïve single goal planning approach.

6.2 Combinatorial Bids

Combinatorial auctions try to capture synergy between goal points (See section 2.3.3). Similarly, the motivation behind coalition forming techniques is to capture synergy between agents. The pairing of combinatorial and coalition forming approaches is worth further study. While assigning goals obtained in a combinatorial fashion to a coalition of agents is essentially a smaller instance of the larger problem of assigning all goals to all agents,

the smaller problem size makes exhaustive approaches more practical once coalitions and goal bundles have been chosen heuristically.

6.3 Advanced Coalition Utility

Combinatorial bid implementations often attempt to bundle goals according to some measure of synergy between them [2]. Similarly, coalition forming techniques could incorporate agent synergy into coalition utility calculations. This may be particularly helpful in extremely heterogeneous systems, where certain types of robots may be particularly well suited for working together. Self-reconfiguring modular robots [11], for example, could benefit greatly by working together, as they may potentially merge to reach otherwise unreachable goals. In other cases, robots may require periodic support from other robots. In such cases, it would be beneficial to pair agents requiring support with agents able to provide support.

6.4 Topological Pattern Matching

While it is impossible to know for sure which goals will branch into multiple new goals to explore, it may be possible to employ pattern matching techniques to predict a likeliness that a goal branches. The ability to predict branching with any accuracy could be conveniently incorporated into a coalition forming exploration strategy. The coalition profit calculation could be easily modified to account for the optimal coalition size for a goal, based on estimated branching. While this would likely be best suited to particularly symmetrical environments, it may nevertheless prove useful in chaotic environments. A similar mechanism could be implemented to make use of low-resolution or unreliable a priori map information, such as images taken by satellite in the case of planetary exploration, to estimate goal utility or branching probability.

Appendices

Appendix A

Result Graphs

This appendix contains results graphs for all the tests performed. The graphs represent the Area Explored Versus Time metric, and are grouped by the exploration environment to which they correspond.

A.1 Open Environment Graphs

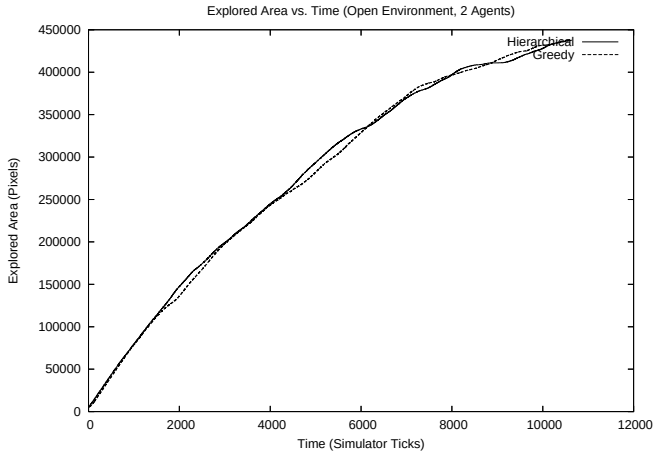

Figure A.1: Area Explored Versus Time graph for 2 agents exploring the open environment

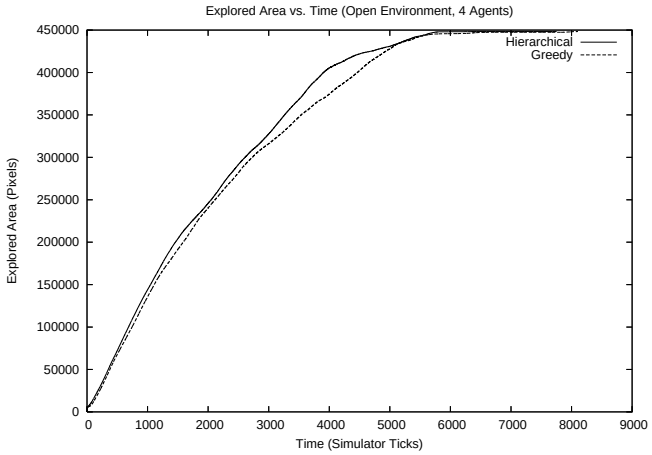

Figure A.2: Area Explored Versus Time graph for 4 agents exploring the open environment

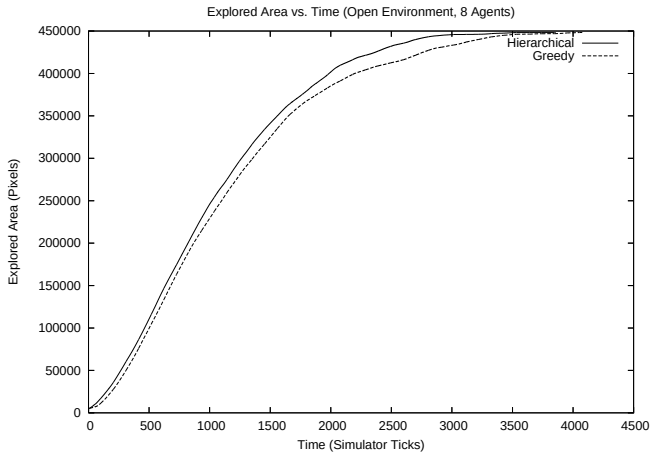

Figure A.3: Area Explored Versus Time graph for 8 agents exploring the open environment

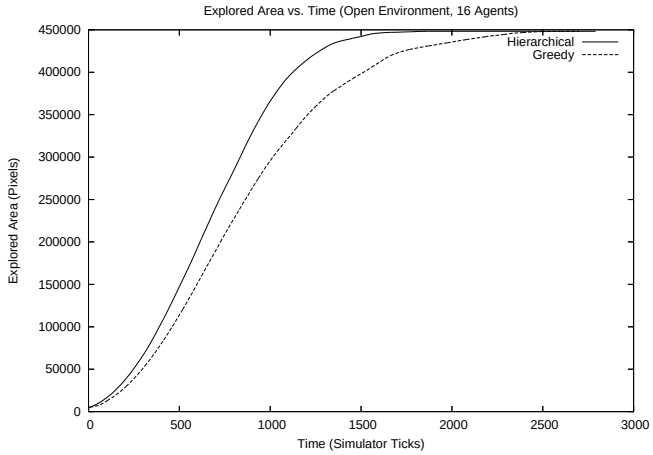

Figure A.4: Area Explored Versus Time graph for 16 agents exploring the open environment

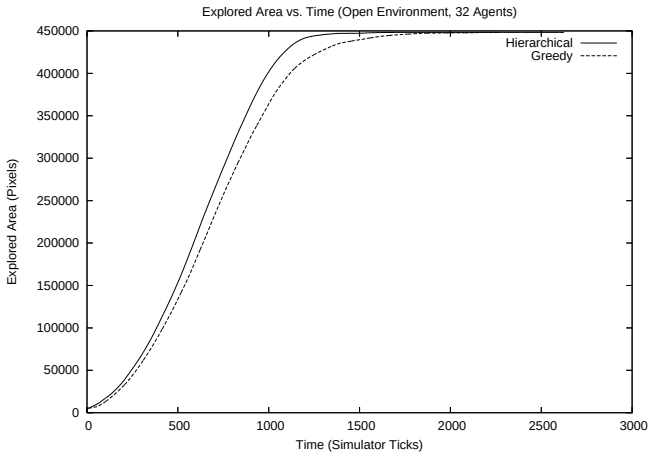

Figure A.5: Area Explored Versus Time graph for 32 agents exploring the open environment

A.2 Sparse Environment Graphs

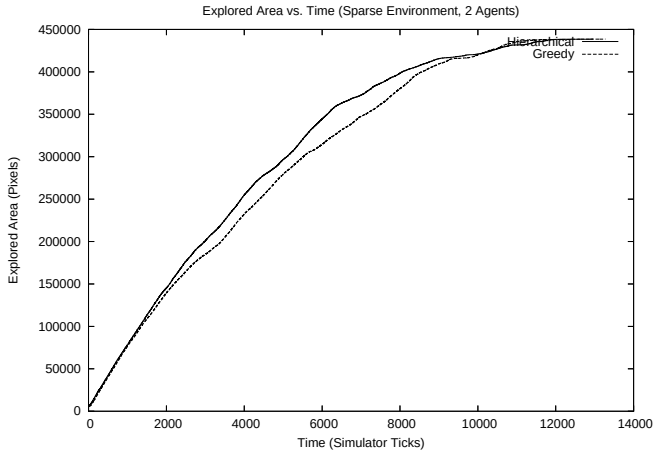

Figure A.6: Area Explored Versus Time graph for 2 agents exploring the sparse environment

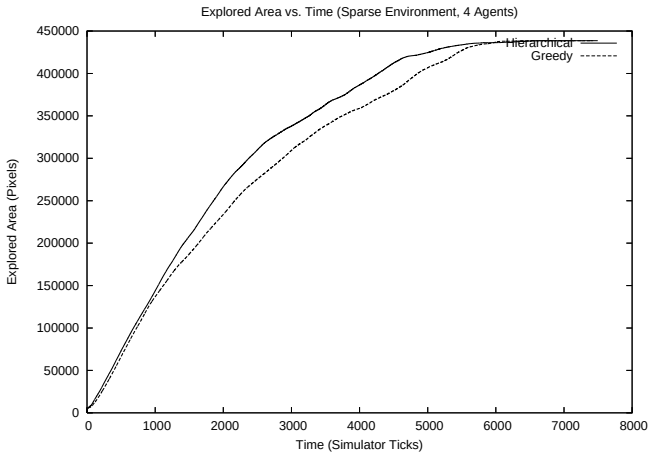

Figure A.7: Area Explored Versus Time graph for 4 agents exploring the sparse environment

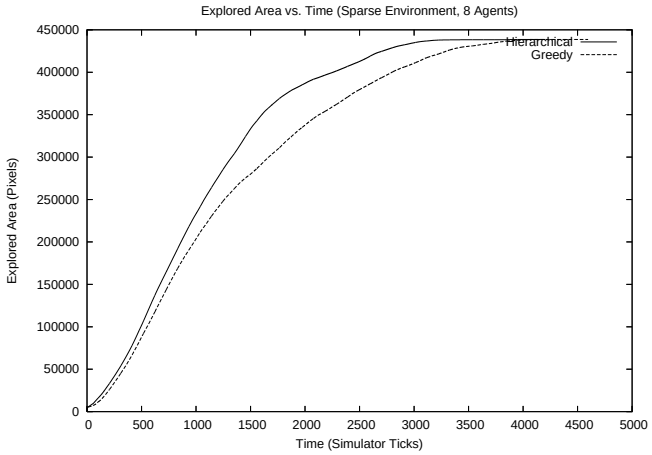

Figure A.8: Area Explored Versus Time graph for 8 agents exploring the sparse environment

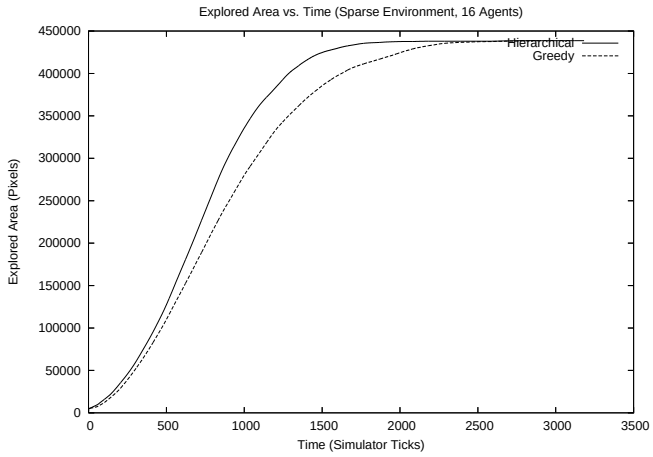

Figure A.9: Area Explored Versus Time graph for 16 agents exploring the sparse environment

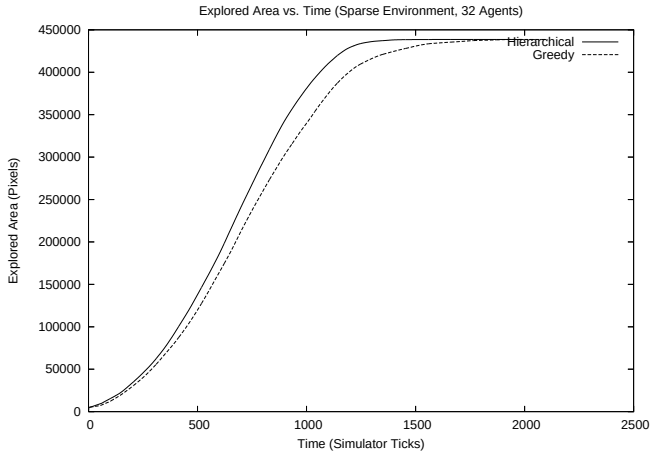

Figure A.10: Area Explored Versus Time graph for 32 agents exploring the sparse environment

A.3 Dense Environment Graphs

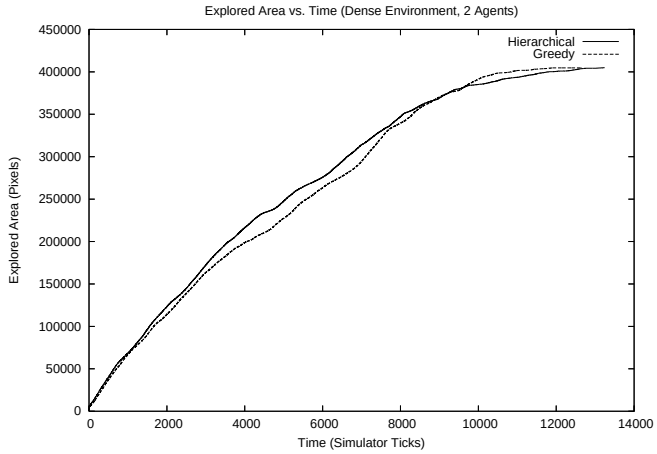

Figure A.11: Area Explored Versus Time graph for 2 agents exploring the dense environment

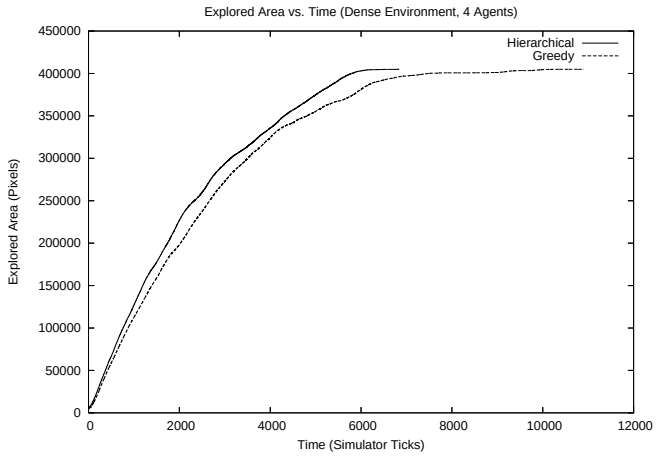

Figure A.12: Area Explored Versus Time graph for 4 agents exploring the dense environment

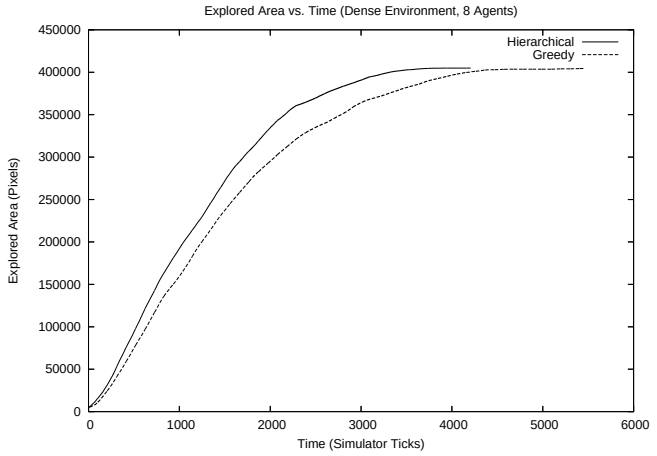

Figure A.13: Area Explored Versus Time graph for 8 agents exploring the dense environment

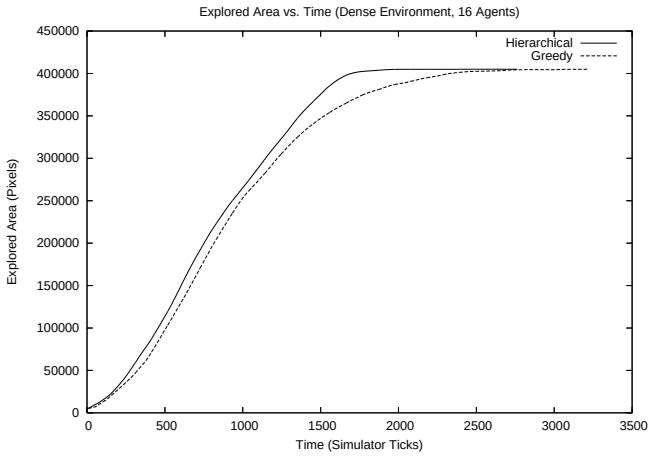

Figure A.14: Area Explored Versus Time graph for 16 agents exploring the dense environment

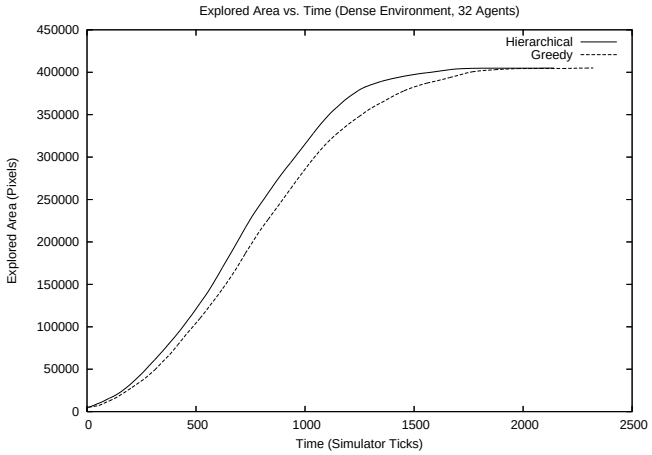

Figure A.15: Area Explored Versus Time graph for 32 agents exploring the dense environment

A.4 Structured Environment Graphs

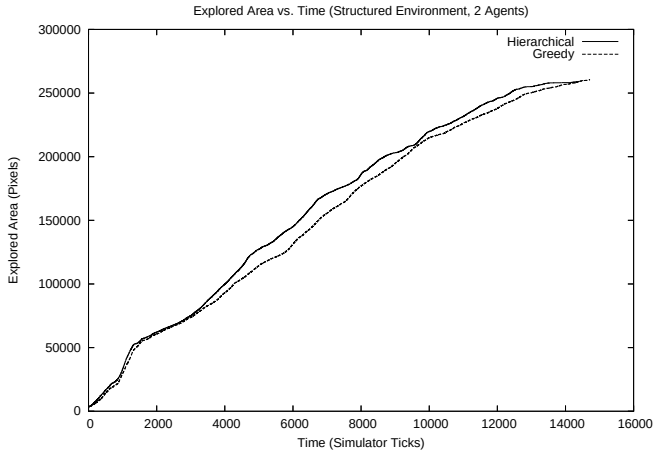

Figure A.16: Area Explored Versus Time graph for 2 agents exploring the structured environment

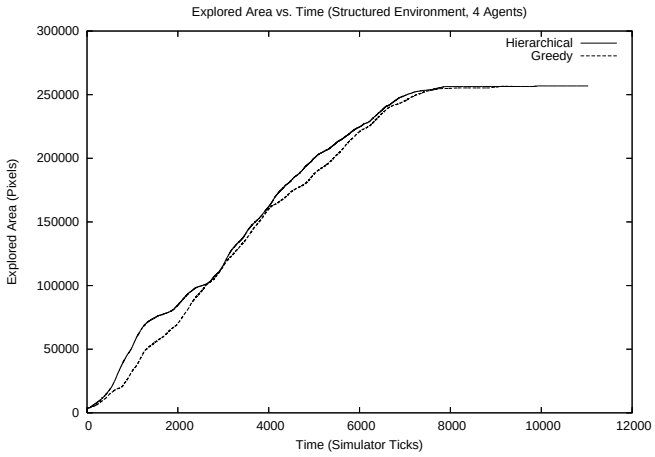

Figure A.17: Area Explored Versus Time graph for 4 agents exploring the structured environment

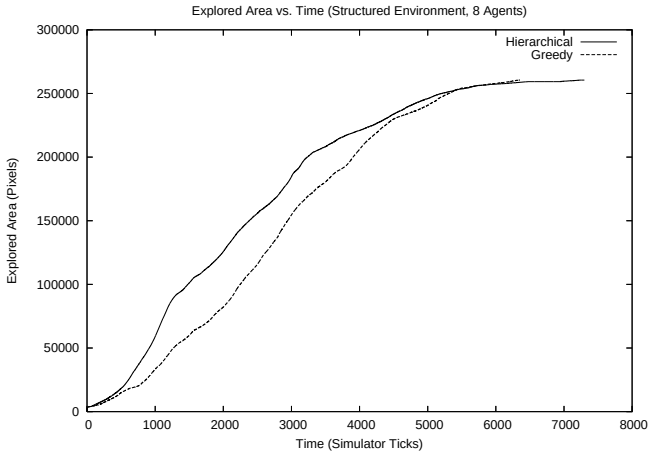

Figure A.18: Area Explored Versus Time graph for 8 agents exploring the structured environment

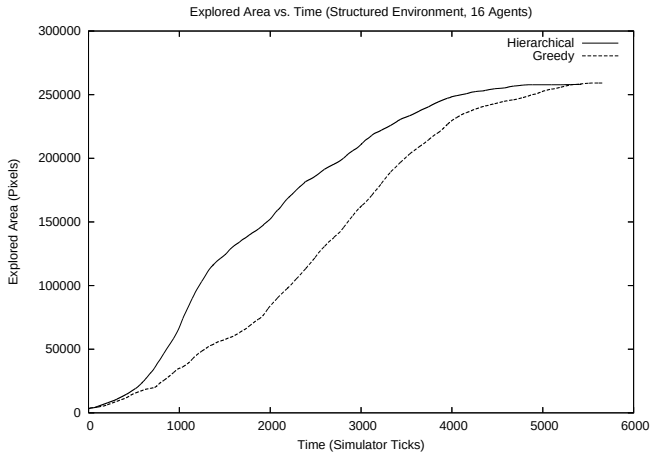

Figure A.19: Area Explored Versus Time graph for 16 agents exploring the structured environment

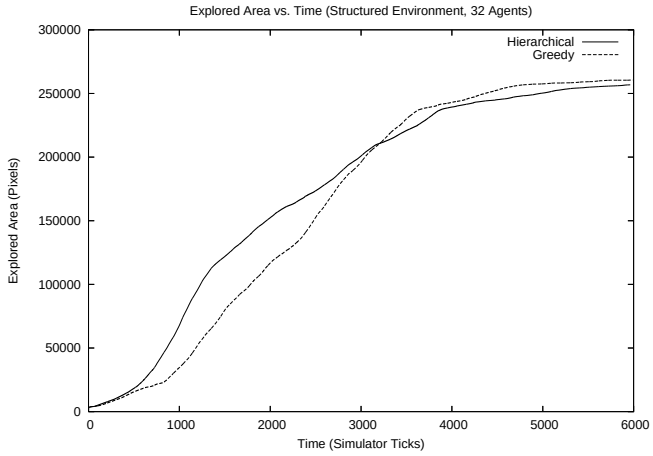

Figure A.20: Area Explored Versus Time graph for 32 agents exploring the structured environment

Bibliography

[1] N.M. Amato and L.K. Dale. Probabilistic roadmap methods are embarrassingly parallel. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1:688–694 vol.1, 1999.

[2] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2:1957–1962 vol.2, Oct. 2003.

[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 1:476–481 vol.1, 2000.

[4] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174, November 1987.

[5] A. Howard and L. Kitchen. Cooperative localisation and mapping. *Proceedings of the 1999 Internation Conference on Field and Service Robotics*, 1999.

[6] L.E. Kavraki, M.N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. *Robotics and Automation, IEEE Transactions on*, 14(1):166–171, Feb 1998.

[7] M.G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A.J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 1:698–705 vol.1, Sept.-2 Oct. 2004.

[8] Liu Lin and Zhiqiang Zheng. Combinatorial bids based multi-robot task allocation method. *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1145–1150, 18-22 April 2005.

[9] Xia Ma, Fang Meng, Yibin Li, Weidong Chen, and Yugeng Xi. Multi-agent-based auctions for multi-robot exploration. *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, 2:9262–9266, 21-23 June 2006.

[10] Xin Ma, Qin Zhang, and Yibin Li. Genetic algorithm-based multi-robot cooperative exploration. *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, pages 1018–1023, May 30 2007-June 1 2007.

[11] Daniela Rus, Zack Butler, Keith Kotay, and Marsette Vona. Self-reconfiguring robots. *Commun. ACM*, 45(3):39–45, 2002.

[12] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.

[13] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. *Proceedings of the National Conference on Artificial Intelligence*, AAAI, 2000.

[14] A. Stentz. Optimal and efficient path planning for partially-known environments. *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317 vol.4, May 1994.

[15] W.E. Walsh and M.P. Wellman. A market protocol for decentralized task allocation. *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 325–332, 3-7 Jul 1998.

[16] C.R. Weisbin and G. Rodriguez. Nasa robotics research for planetary surface exploration. *Robotics & Automation Magazine, IEEE*, 7(4):25–34, Dec 2000.

[17] B. Yamauchi. A frontier-based approach for autonomous exploration. *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151, 10-11 Jul 1997.

[18] Brian Yamauchi. Frontier-based exploration using multiple robots. In *AGENTS '98: Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53, New York, NY, USA, 1998. ACM.

[19] R. Zlot, A. Stentz, M.B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 3:3016–3023, 2002.