

UNIVERSITY of CALIFORNIA
Santa Barbara

**Graph-Based Approximation Algorithms for Cooperative Routing
Problems**

A Dissertation submitted in partial satisfaction of the
requirements for the degree

Doctor of Philosophy

in

Electrical and Computer Engineering

by

James Robert Riehl

Committee in charge:

Professor João P. Hespanha, Chair

Professor Francesco Bullo

Professor John R. Gilbert

Professor Roy S. Smith

August 2007

The dissertation of James Robert Riehl is approved.

Professor Francesco Bullo

Professor John R. Gilbert

Professor Roy S. Smith

Professor João P. Hespanha, Committee Chair

July 2007

Graph-Based Approximation Algorithms for Cooperative Routing Problems

Copyright © 2007

by

James Robert Riehl

Curriculum Vitæ

James Robert Riehl

Education

- 2002 B.S. Engineering, Harvey Mudd College, Claremont, California.
- 2004 M.S. Electrical Engineering, University of California, Santa Barbara.

Experience

- 2003–2004 Teaching Assistant, University of California, Santa Barbara.
- 2004–2007 Research Assistant, University of California, Santa Barbara.

Selected Publications

J. Riehl, J. Hespanha, “Fractal Graph Optimization Algorithms.” In *Proceedings of the 44th IEEE Conference on Decision and Control*. December, 2005.

B. DasGupta, J. Hespanha, J. Riehl, E. Sontag. “Honey-pot Constrained Searching with Local Sensory Information.” *Nonlinear Analysis: Hybrid Systems and Applications*. Volume 65, Number 9, pp 1773–1793, November, 2006.

J. Riehl, J. Hespanha, “Cooperative Graph Search Using Fractal Decomposition.” In *Proceedings of the 2007 American Control Conference*. July, 2007.

J. Riehl, J. Hespanha. “Graph Optimization Using Fractal Decomposition with Application to Cooperative Routing Problems.” To be submitted to journal publication.

J. Riehl, G. Collins, J. Hespanha, “Cooperative Graph-Based Model Predictive Search.” To be presented at the *46th IEEE Conference on Decision and Control*. December, 2007.

Abstract

Graph-Based Approximation Algorithms for Cooperative Routing Problems

by

James Robert Riehl

This work focuses on finding good approximations to large scale, computationally complex cooperative routing problems. These optimizations may be defined on a graph, or a continuous domain, on which we construct a graph. In the latter case, we show that the construction of the graph is a key step in the approximation algorithm and is vital to its performance. The remainder of this work is a combination of two differing approaches to the approximation of cooperative routing graph optimizations. In each case, the goal is the same: find fast approximation algorithms that generate close-to-optimal solutions.

The first method, which we call fractal decomposition, works by recursively partitioning the graph in order to reduce the original large problem to several smaller problems. The solutions to these smaller problems are used to construct bounds on the optimal solution and generate an approximate solution. This general method applies to a wide range of graph optimizations for which the data is static, including shortest path, maximum flow, and graph search. As the number of decomposition levels increases, the computational complexity approaches $O(n)$ at the expense of looser approximation bounds. We demonstrate the fractal decomposition method on three example problems related to cooperative routing: shortest path matrix, maximum flow matrix, and cooperative search. Large-scale

simulations show that this fractal decomposition method is computationally fast and can yield good results for practical problems.

The second approximation method is a receding horizon approach applied to a search problem, in which multiple agents are cooperatively searching for a mobile target in a bounded continuous region. By sampling the region of interest at locations with high target probability, we reduce the continuous search problem to an optimization on a finite graph. Paths are computed on this graph using a receding horizon approach, in which the horizon is a fixed number of waypoints. To facilitate a fair comparison between paths of varying length on a non-uniform graph, we use an optimization criterion corresponding to the probability of finding the target per unit time. Using this algorithm, we show that the team discovers the target in finite time with probability one. Simulations show that this algorithm makes efficient use of agents and performs quite well compared to previously proposed cooperative search algorithms.

Contents

Curriculum Vitæ of James Robert Riehl	iv
Abstract	vi
List of Figures	xi
1 Introduction	1
1.1 Cooperative Search Theory	3
1.2 Organization and Summary of Results	7
2 Graphs	11
2.1 Optimization on Graphs	12
2.1.1 Shortest path matrix	12
2.1.2 Maximum flow matrix	13
2.1.3 Graph search	13
2.1.4 Cooperative graph search	14
2.2 Graph Partitioning	14
2.2.1 Partitioning Algorithms	15
3 Continuous to Discrete: Graph Generation by Sampling	18
3.1 A Continuous Shortest Path Problem	19
3.1.1 Problem Formulation and Background	20
3.1.2 Local Discretization	24

3.1.3	Computational Complexity	29
3.1.4	Non-uniform Sampling	31
3.1.5	Minimum-Risk Path Planning for Groups of UAVs	35
4	Fractal Decomposition	47
4.1	Related work	49
4.2	Method	51
4.2.1	Partition the graph	51
4.2.2	Construct bounding meta-problems and solve	52
4.2.3	Refine worst-case solution to approximate solution on original graph	53
4.3	Computational Complexity	53
4.4	Shortest Path	56
4.4.1	Problem Formulation	56
4.4.2	Approximation Error Bounds for Lattice Graphs	61
4.4.3	Case Study on Shortest Path Routing	63
4.5	Maximum Flow	65
4.5.1	Problem Formulation	66
4.5.2	Approximation Error Bounds for Lattice Graphs	71
4.5.3	Case Study on Stochastic Max-Flow Routing	74
4.6	Cooperative Graph Search	75
4.6.1	Problem Formulation	76
4.6.2	Tightness of Approximation for Lattice Graphs	84
4.6.3	Cooperative Search Test Results	86
5	Graph-Based Receding Horizon Optimization	90
5.1	Method	91
5.2	Search Problem Formulation	92
5.2.1	Example 1: Particle Filter	96
5.2.2	Example 2: Grid-based Probabilistic Map	97
5.2.3	Problem Statement	98

5.3	Cooperative Graph-based Model Predictive Search Algorithm . . .	100
5.3.1	Graph Search Formulation	100
5.3.2	Dynamic Graph Generation	104
5.3.3	CGBMPS Algorithm	108
5.3.4	Computational Complexity	110
5.4	Persistent Search	111
5.5	Simulations	113
6	Conclusion	116
A	Proofs	120
	Bibliography	127

List of Figures

2.1	Drawing of a graph.	11
2.2	Example of a graph partitioned into 4 <i>meta-vertices</i>	15
3.1	Honeycomb sampling	33
3.2	Minimum-risk path planning scenario.	38
3.3	Paths computed for this scenario using various sampling algorithms.	40
3.4	Three dimensional view of path.	41
3.5	Path cost vs. computation time.	42
3.6	Cubic-spline path smoothing	46
3.7	Three-dimensional view of smoothed path	46
4.1	Example of recursive decomposition on a clustered graph (edges not shown).	47
4.2	The left figure shows an overhead view of obstacles in a region. The graph to the right is generated from sampling where the gradient of the obstacle map is high, and connecting nodes with a Delaunay triangulation.	64
4.3	Example showing the worst-case edge-cost between two meta-vertices. Edge-costs are 1 for edges within subgraphs, 2 for edges between subgraphs and all vertex costs are 0.	80
4.4	Model of third floor of UCSB’s Harold Frank Hall divided into 646 cells and overlaid with a graph. Dark cells indicate high target probability.	86
4.5	Two levels of partitioning on the search graph.	88

4.6	Results of 4-agent cooperative search simulation.	88
5.1	Diagram of a field of regard (FOR) and a field of view (FOV). . .	92
5.2	Example sensor schedule for an agent.	101
5.3	Example of graph construction process.	106
5.4	Display of Search Simulation.	113

Chapter 1

Introduction

Graphs are used to solve problems in almost every field of engineering, and as the problems we want to solve become more complex and computers more powerful, the use of graphs is ever increasing. However, for very large scale graph problems, especially problems that are computationally complex to begin with, optimal solutions can still be intractable. When facing such obstacles in computational complexity, it is useful to develop fast approximation algorithms, and ideally, guarantee some bounds on the resulting approximation error. This dissertation focuses on two approximation methods for graph optimization problems: *fractal decomposition*, and *receding horizon*.

Before introducing these approaches, it is important to discuss the origin of the graphs on which the optimization is performed. If the problem is already defined on a discrete domain, generating the graph is straight-forward, but cooperative routing problems are often defined on a continuous domain. When the continuous optimization problem is computationally intractable, as it is in the cooperative

search problem for example, one can approximate the solution by solving an optimization on a finite graph. However, the method used to construct this graph will have a major effect on the computation time required by the algorithm as well as the resulting approximation error. One of the goals of this dissertation is to provide a methodology for generating graphs that result in close-to-optimal solutions with relatively low computation times.

In some cases, the resulting graph optimization problem may be solved in a reasonable amount of time by using standard algorithms. If not, we may be faced with a discrete problem that is still computationally too complex to solve directly. The idea encompassing both fractal decomposition and receding horizon methods is that we can reduce the complexity of a problem by breaking it down into smaller, more manageable parts. The fractal decomposition method achieves this through spatial decomposition, by partitioning a graph representing the state space of the problem, and solving several smaller problems. It then uses the solutions to the smaller problems to construct an approximate solution to the original problem and generate bounds on the resulting approximation error. On the other hand, the receding horizon can be viewed as a kind of temporal decomposition, in which one approximates the solution to a problem by repeatedly solving a series of shorter, finite horizon optimization problems.

In this work, we specifically consider optimization problems related to cooperative routing of multiple autonomous agents. Types of missions in which these problems occur include

1. Minimum time routing for a group of agents through a region with obstacles
2. Multiple-agent routing while avoiding detection or interception

3. Sending many agents through a network of routes with limited capacity
4. Searching for one or more targets with uncertain location

The first mission is easily formulated as a shortest path problem on a graph generated by sampling the region of interest. The second may be solved by a shortest path problem or a maximum flow problem depending on the nature of the adversary. The third mission is a direct application of the maximum flow problem. Finally, we can regard mission four as a graph search problem. Each of the above missions is assigned to a team of agents, who may need to cooperate to achieve the best possible performance. Detailed examples of these problems are presented throughout the remainder of this dissertation with particular emphasis on the cooperative search problem, as it poses the biggest challenge with respect to computation.

1.1 Cooperative Search Theory

A large portion of this work focuses on the interesting and computationally complex class of problems that can be categorized as cooperative search. These are problems in which multiple mobile agents with limited sensing range search a region for one or more objects whose locations are unknown or uncertain. During the search, they incur some cost, whether it be time, energy, or some other quantity. The goal of each individual agent then is to search in a way that results in the *team* finding the object(s) with minimum cost, or with maximum probability subject to a constraint on the cost.

Modern search theory was pioneered by the work of Koopman [19], Stone

[28], and others, whose initial motivation was to develop efficient search methods to find enemy marine vessels. More recently, agencies such as the U.S. Coast Guard have applied search theory to search and rescue missions with great success measured in saved lives [9]. Other notable applications of search theory include exploration, mining, and surveillance [13].

Early search theory focused on the allocation of search effort to specific areas within the search region, which is appropriate to special cases for which the searcher motion is unconstrained, or when finding optimal search paths on these areas is otherwise intuitive. For example, suppose that we have a stationary target and we can narrow down its location to a rectangular region over which there is a uniform probability distribution. Furthermore, the searchers have perfect sensors, that is, they detect the target with probability one if it is in their field of view and return no false detections. Then, there is no better search path than a lawnmower type pattern such as in [7], and the problem reduces to determining which rectangular regions to search. For an alternate example, consider a stationary agent that can illuminate a subset of the search region with a beam of light and search that subset. Moving the light beam takes very little time compared to searching the illuminated region. We have now reduced the problem to selecting which regions to illuminate. This dissertation focuses on *constrained search* problems, where we are presented with the more difficult task of finding optimal paths for the searchers.

Mangel formulated the continuous constrained search problem as an optimal control problem on the searcher's velocity subject to a constraint in the form of a partial differential equation, but this problem is only solvable for very simple initial target probability distributions [22]. A more practical approach is to dis-

cretize the search space and formulate the search problem on a graph. Under such a formulation, the set of search paths is restricted to piecewise linear paths connecting a finite set of graph vertices. Although this discretization simplifies the problem to some extent, it is still computationally difficult. Trummel and Weisinger showed that the single-agent search problem on a graph is NP-Hard even for a stationary target [30]. Eagle and Yee [6] were able to solve somewhat larger problems than what had previously been possible by formulating the problem as a nonlinear integer program and using a branch and bound algorithm to solve it. However, the size of computationally feasible problems was still severely limited.

The problem gains additional complexity when we consider multiple agents that are cooperatively searching for a target. For this reason, literature on cooperative search generally seeks approximate solutions.

One way to reduce the size of an optimization problem and thus the computation time is to partition the state-space and solve a set of smaller problems. For the search problem, this generally translates to dividing the search region into smaller subregions. Then one can find the optimal search path on each subregion and piece them together, ideally providing bounds on the resulting approximation error. DasGupta *et al.* presented one such approach to the stationary target search based on an aggregation of the search space using a graph partition [5]. Here, we expand on this idea by allowing for multiple levels of decomposition and multiple cooperating agents. In the cooperative search problem, if we account for all possible configurations of agents, the state-space is much larger. An expanded graph representing all these states would have n^M vertices, where n is the number of vertices in the original graph, and M is the number of agents. To further

reduce the size of this problem, in addition to partitioning the search space into subregions, we also enforce a maximum occupancy, allowing only a subset of the agents to search a single subregion at the same time. Currently, this method only applies to the search for stationary targets.

When the target is mobile, an intuitive way to reduce the computation involved in finding optimal search paths is to restrict the optimization to a finite, receding horizon. Somewhat surprisingly, even very short optimization horizons can yield good results. Hespanha *et al.* [15] showed that in pursuit-evasion games played on a discrete grid, one step Nash policies result in finite-time evader capture with probability one. Using slightly longer horizons, Polycarpou *et al.* introduced a finite-horizon look-ahead approach similar to model predictive control as part of a cooperative map learning framework [24]. Chung *et al.* also included a receding horizon search algorithm in [29] as part of a decision strategy for probabilistic search. A challenge faced by these receding horizon algorithms is that because they require the solution to an NP-hard search problem at each time step, the prediction horizon must be kept short to ensure computational feasibility. This means that if there are regions of high target probability separated by a distance that is much greater than an agent can cover on this horizon, the algorithm’s performance will suffer. One does not have to produce a pathological example to generate such a situation, because even a uniform initial distribution can evolve into a highly irregular distribution as the searchers move through the region and update their estimates based on incoming sensor measurements. In the Cooperative Graph-Based Model Predictive Search (CGBMPS) algorithm of Chapter 5, we address this issue by optimizing over a dynamically changing graph whose nodes are carefully placed at locations in the search region having

the highest target probability.

1.2 Organization and Summary of Results

Chapter 2 gives a brief overview on graphs, graph optimization problems, and graph partitioning, introducing notation and terminology that we will use throughout the remainder of this dissertation.

In Chapter 3, we discuss how to convert an optimization on a continuous domain to one on a discrete, finite graph by performing a selective non-uniform sampling of the continuous region. This chapter specifically addresses the continuous anisotropic weighted shortest path problem, in which the cost associated with a path depends not only on its length but also in the instantaneous direction of motion. For this problem, we sample along the boundaries of a convex partition of the region of interest, generating a “Honeycomb” sampling pattern. In Section 3.1.2, we prove that the cost of paths generated using this technique are guaranteed to be within ϵ of the minimum path cost, where ϵ can be made arbitrarily small. This algorithm is a computational improvement on a similar algorithm proposed in [18]. Simulations show that this method indeed finds low-cost paths with low computation compared to previous algorithms. Furthermore, we provide a cubic-spline smoothing algorithm that takes the piecewise straight line path and generates a trajectory with bounded acceleration to accommodate dynamical constraints of a UAV. In many cases, this algorithm also results in an improvement in the path cost.

In Chapter 4, we introduce the fractal decomposition method for approximating the solution to graph optimization problems. Section 4.1 discusses pre-

existing literature on the hierarchical decomposition of various graph problems. In Section 4.2, we outline the general procedure for implementing the fractal decomposition method, and Section 4.3 presents the computational complexity of the resulting algorithm as a function of the computational complexity of the original problem. The qualifier “fractal” refers to the fact that the original problem is decomposed into several smaller problems of the same type, each of which could be further decomposed using the same technique. In Sections 4.4, 4.5, 4.6, we present results of the following form for all-pairs shortest path, all-pairs maximum flow, and cooperative graph search, respectively.

Let J_G^* represent the optimal value for a given graph optimization problem (*e.g.* minimum path cost, maximum flow intensity, maximum search reward). Given a graph $G := (V, E)$ and any partition \bar{V} on that graph, we provide a procedure to construct meta-graphs \bar{G}_{worst} and \bar{G}_{best} such that J_G^* is bounded above and below by $J_{\bar{G}_{\text{worst}}}^*$ and $J_{\bar{G}_{\text{best}}}^*$, which are obtained by performing a similar optimization on the meta-graphs \bar{G}_{worst} and \bar{G}_{best} , respectively.

Furthermore, we give problem-independent approximation error bounds for certain regular graphs such as lattices. In Section 4.4.2, for the shortest path problem applied to d -dimensional lattice graphs, we show that the worst-case approximation bound on the diameter is within a factor of d^L of the actual diameter, where L is the number of decomposition levels used.

In Section 4.5.2, for the maximum flow problem applied to d -dimensional lattice graphs, we show that the worst-case bandwidth approximation is equal to the actual bandwidth. In the case of *toroidal* lattices, lattice graphs that wrap around like a torus, the worst-case bandwidth approximation is equal to $\frac{1}{2}$ the actual bandwidth.

In Section 4.6.2, for the graph search problem on a lattice graph with unit edge costs, unit reward at each vertex, and a cost bound $L = \gamma n$, where n is the number of vertices, we show that the worst-case search reward is an asymptotically tight bound on the optimal search reward as n approaches infinity.

For the bounds $J_{\bar{G}_{\text{worst}}}^*$ and $J_{\bar{G}_{\text{best}}}^*$ to be useful, they must be achievable with significantly less computation than what is required to solve the original problem. Indeed, in section 4.3 we show that the computational complexity required for the fractal decomposition method approaches $O(n)$ as the number of decomposition levels increases. The penalty to be paid for this savings in computation is that the approximation bounds become further apart.

For the three problems mentioned above (all-pairs shortest path, all-pairs maximum flow, and cooperative graph search), we provide simulations based on cooperative routing scenarios in Sections 4.4.3, 4.5.3, and 4.6.3, respectively. In each case, we show that fairly large-scale problems can be quickly approximated and the resulting solutions lie within a relatively small range of the optimal criteria for the original problem.

Chapter 5 presents the results on our receding horizon approach to the cooperative search for a mobile target. Section 5.1 summarizes the method, Section 5.2 formulates the problem, and Section 5.3 provides a detailed description of the Cooperative Graph-Based Model Predictive Search (CGBMPS) algorithm. In Section 5.4, we show that a cooperating team of agents employing this algorithm will achieve finite time target discovery with probability one. Simulations in Section 5.5 show that this algorithm makes good use of search assets and performs significantly better than previously proposed cooperative search algorithms. The CGBMPS algorithm has also been successfully tested on a hardware platform

consisting of two Unmanned Aerial Vehicles (UAVs) with gimbal-mounted cameras.

Chapter 6 summarizes the results and contributions of this work, with several suggestions for promising directions of future research on these topics.

Chapter 2

Graphs

The graph is a mathematical tool that has proven useful in a wide range of engineering applications. Particular, it is convenient to represent any structure or system having discrete states and transitions between them with a graph. Although graphs may be used to represent systems with complex discrete dynamics, they are more commonly used in situations where the dynamics are simple, but the structure of the transitions is complex or the scale is very large. Additionally, one can use a graph to approximate the states of a continuous-space system with the use of sampling. This is discussed further in Chapter 3.

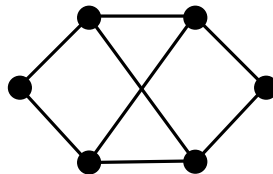


Figure 2.1. Drawing of a graph.

We denote a graph by $G := (V, E)$, where V is a set of *vertices* and $E \subseteq V \times V$ is a set of *edges* connecting pairs of vertices. Figure 2.1 shows an example drawing

of a graph with six vertices and eight edges.

2.1 Optimization on Graphs

One of the primary uses for graphs in engineering is in optimization problems. For example, given a graph whose vertices represent locations in space and whose edges determine passageways between them, we may want to find the shortest path between each pair of vertices. This involves assigning a cost to each edge indicating the distance between its endpoint vertices. Then the problem may be solved by various standard algorithms.

We now introduce four graph optimization problems along with the best known computational complexity results. In what follows, we use the standard notation $O(f(n, m))$ to denote the class of functions that are bounded above by a constant factor of $f(n, m)$. In other words, $g(n, m) = O(f(n, m))$ if and only if there exists a constant $\alpha > 0$ such that $|g(n, m)| < \alpha f(n, m)$ for all values of n and m .

2.1.1 Shortest path matrix

The shortest path matrix problem, also called all-pairs shortest paths, involves finding the minimum-cost path between every pair of vertices in a graph. There are a great number of applications of this problem, including dynamic programming [27], and optimal route planning for groups of agents [18]. For a weighted directed graph with n vertices and m edges, Karger *et al.* showed that the all-pairs shortest paths problem can be solved with computational complexity

$O(nm + n^2 \log n)$ [17].

2.1.2 Maximum flow matrix

The maximum flow matrix problem involves finding, for each pair of vertices in a graph, the flow assignment on the edges that yields the maximum flow intensity, subject to capacity constraints on the edges. Applications of this problem include stochastic network routing [21] and vehicle routing [4]. Given a directed graph $G(V, E)$ with n vertices and m capacitated edges, Goldberg and Tarjan [11] showed that one can compute the maximum flow between two vertices in $O(nm \log \frac{n^2}{m})$ time. To compute the maximum flow between all pairs of vertices in an *undirected* graph, Gomory and Hu showed that one only needs to solve $n - 1$ maximum flow problems [12], but for directed graphs, we must compute the flow between all $\frac{n(n-1)}{2}$ vertices. This results in a complexity of $O(n^3 m \log \frac{n^2}{m})$ to generate the complete maximum flow matrix.

2.1.3 Graph search

In the graph search problem, there is a cost associated with each edge, typically corresponding to a quantity of time or energy spent in transit along that edge, and a reward associated with each vertex, generally corresponding to the probability of a hidden target being located at that vertex. The objective of the graph search problem is to find the path in a graph that maximizes the probability of finding the target, subject to a constraint on the path cost. In [30], the computational complexity of the graph search problem for a single agent was shown to be NP-complete on the number of vertices n .

2.1.4 Cooperative graph search

The cooperative graph search problem is an extension of the graph search problem for a team of cooperating agents. The objective is to find optimal paths in the graph that maximize the probability that the team will find the target, subject to a cost constraint on the paths of the individual agents. Reducing the M -agent cooperative search problem to a single-agent search problem with n^M vertices results in a problem that is also NP-complete. In the worst case, an exhaustive search on a complete graph would have complexity $O(n^M!)$. Although there are some more efficient algorithms to solve this problem such as the branch and bound methods of Eagle and Yee [6], this problem is still computationally infeasible for large values of n and M .

2.2 Graph Partitioning

The examples given above range in computational complexity from being manageable for a large number of nodes, as in the shortest path matrix problem, to being infeasible for even for a relatively small scale problem, as in the cooperative graph search problem. In Chapter 4, we will present a method for reducing the computational complexity associated with these problems, at the expense of some sub-optimality, by using *graph partitioning*.

Given a graph $G := (V, E)$, a *partition* $\bar{V} := \{\bar{v}_1, \bar{v}_2, \dots, \bar{v}_k\}$ of G is a set of disjoint subsets of V such that $\bar{v}_1 \cup \bar{v}_2 \cup \dots \cup \bar{v}_k = V$. We call these subsets \bar{v}_i *meta-vertices*. For a given meta-vertex $\bar{v}_i \in \bar{V}$, we define the *subgraph of G induced by \bar{v}_i* to be the graph consisting of the vertices $v \in \bar{v}_i$ and the edges connecting pairs

of those vertices, and we denote this subgraph by $G|\bar{v}_i := (\bar{v}_i, E \cap \bar{v}_i \times \bar{v}_i)$.

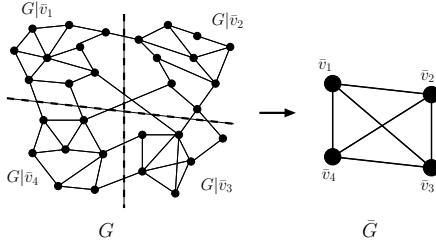


Figure 2.2. Example of a graph partitioned into 4 *meta-vertices*.

Figure 2.2 shows an example of the graph partitioning process, where the dashed lines through G separate the partitioned subgraphs $G|\bar{v}_i$, which are represented by meta-vertices \bar{v}_i in \bar{G} .

Given a partition \bar{V} of the vertex set V , we define the *meta-graph of G induced by the partition \bar{V}* to be the graph $\bar{G} := (\bar{V}, \bar{E})$ with an edge $\bar{e} \in \bar{E}$ between meta-vertices $\bar{v}_i, \bar{v}_j \in \bar{V}$ if and only if G has at least one edge between vertices v and v' for some $v \in \bar{v}_i$ and $v' \in \bar{v}_j$. In general, there may exist several such edges $e \in E$ and we call these the *edges associated with the meta-edge \bar{e}* .

2.2.1 Partitioning Algorithms

It is convenient to perform graph partitions using an automated algorithm because this allows one to incorporate the partitioning process into the approximation algorithms for various graph optimization problems without requiring external inputs or manipulation. In this section, we discuss some challenges involved in graph partitioning, present a few alternative approaches, and describe the specific method we use for the examples in Chapter 4.

Our main objective will be to generate partitions that maximize the accu-

racy of approximation while keeping the computation low, or possibly minimize computation while keeping a certain level of accuracy. This is one of the fundamental challenges of graph partitioning. In an attempt to approximately solve this problem, we will preprocess the graph data in way that facilitates a partition satisfying our objectives. For example, in the shortest path problem, we achieve tighter approximation bounds if the maximum distance between vertices in the same meta-vertex is small compared to the distance between vertices in neighboring meta-vertices. This suggests using an algorithm that clusters vertices that are graphically close. Alternatively, in the maximum flow problem, the best bounds result when the lowest maximum flow between two vertices in the same meta-vertex is large compared to that between vertices in neighboring meta-vertices. Hence, the partitioning algorithm should be chosen carefully to achieve the goals of the specific problem.

Various ways of partitioning graphs have been proposed in the literature. See [8] for an extensive survey of these results. Many graph partitioning schemes can be categorized as *geometric* or *spectral*. Geometric algorithms partition a graph based on the spatial location of its vertices, whereas spectral algorithms use eigenvector analysis on a matrix representation of a graph, such as the adjacency matrix or Laplacian matrix.

For the examples in Sections 4.4.3, 4.5.3, and 4.6.3, we use the automated spectral graph partitioning algorithm introduced in [14]. This algorithm is designed to be an approximation to the l -bounded graph partitioning problem, which is the problem of finding the graph partition that minimizes the total cost of cut edges subject to the constraint that no element of the partition contains more than l vertices. The technique used in this algorithm is to cluster the eigen-

vectors of a doubly stochastic modification of the edge-cost matrix around the k most linearly independent eigenvectors. The vertices corresponding to these eigenvector clusters are grouped together resulting in a k -partition of the graph. Note that one can design the edge cost matrix used by this algorithm to suit a particular problem. For example, in the cooperative graph search example of Section 4.6.3, we define the edge costs between adjacent vertices with rewards r_1 and r_2 to be $e^{-|r_1-r_2|}$. This causes the algorithm to favor cutting edges with very different rewards, and thus grouping vertices with similar rewards.

Chapter 3

Continuous to Discrete: Graph Generation by Sampling

Many applications of graph optimization arise from problems on a continuous domain. This is largely because, when dealing with computationally difficult problems, it is often more practical to optimize on a discrete graph than a continuum. Solutions to the discrete graph problem must then be refined into solutions to the continuous problem, which are likely to be suboptimal. However, it may be possible to construct bounds on the resulting approximation error. In this chapter, we show how to use sampling to approximate a continuous shortest path problem to within ϵ of the optimal solution, where ϵ can be made arbitrarily small. A key step in this type of problem is generating a graph that results in good approximations to the continuous problem. With a uniform sampling of the state space, more samples will lead to better approximations, but there is a tradeoff here, because increasing the number of samples, and thus graph vertices,

will generally increase computation. We explore this issue in Section 3.1.4, and propose a non-uniform sampling technique that still yields ϵ -optimal solutions with relatively few samples, and therefore low computational complexity.

3.1 A Continuous Shortest Path Problem

This section presents a discrete approximation to a generalized shortest path problem on a continuous domain. The goal is to find the path between two points that minimizes the line integral of a cost-weighting function along the path. Our approach is inspired by the work in [18], where Kim and Hespanha approached this problem with a “Honeycomb” sampling technique. In this paper, the authors show how to construct a graph on which the cost of the optimal path is within ϵ of the minimum path-cost for the continuous problem. However, the proposed algorithm uses a highly connected graph, and depending on how tight an approximation is desired, the optimization may still require a large amount of computation. Here, we will improve upon these results by constructing a graph that is fully connected only within each element of a convex partition of the search region. As shown in Section 3.1.3, this significantly reduces computation with no loss in optimality compared to the algorithm in [18]. We also provide a theoretical foundation for using the “Honeycomb” sampling method because in Section 3.1.2, we state a theorem that proves the existence of ϵ -optimal paths connecting points on the boundaries of a convex partition of the region of interest. Section 3.1.4 gives a detailed description of the non-uniform Honeycomb sampling algorithm, and the simulations in Section 3.1.5 show that this algorithm compares favorably to other sampling algorithms by computing paths of the same level of optimality

with significantly reduced computation time. Furthermore, Section 3.1.5 shows how to use the resulting piecewise straight-line paths to construct cubic spline solutions that respect the dynamical constraints of an agent traveling along the path.

3.1.1 Problem Formulation and Background

Since we are considering exactly the same problem as in [18], the following problem formulation is directly based on the one given in that paper.

Consider two points $x_i, x_f \in \mathcal{R}$ in a compact region $\mathcal{R} \subset \mathbb{R}^n$. Our goal is to compute a continuous path ρ from x_i to x_f that minimizes the line integral over ρ of a cost-weighting function ℓ that depends on the position and direction of motion. This is often referred to as a *shortest-path* optimization. To emphasize the fact that the cost-weighting is not uniform and that it depends on the direction of motion, we further qualify it as a *weighted anisotropic shortest-path* optimization.

Let \mathcal{P} denote the set of all unit-speed paths in \mathcal{R} from x_i to x_f that are continuous and piecewise twice continuously differentiable, *i.e.* the set of continuous functions $\rho : [0, T] \rightarrow \mathcal{R}$, for which

1. $\rho(0) = x_i$ and $\rho(T) = x_f$
2. $\dot{\rho}, \ddot{\rho}$ exist on $[0, T]$, except for a finite number of points
3. $\|\dot{\rho}\| = 1$ wherever this derivative exists.

We formalize the problem under consideration as follows:

Problem 1: Weighted anisotropic shortest-path. Compute a path $\rho^* \in \mathcal{P}$ such that

$$J[\rho^*] = \inf_{\rho \in \mathcal{P}} J[\rho],$$

where $J : \mathcal{P} \rightarrow [0, \infty)$ denotes the cost functional defined by

$$J[\rho] := \int_0^T \ell(\rho(t), \dot{\rho}(t)) dt,$$

for each $\rho : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} . □

Throughout this section, we assume that the cost-weighting function $\ell : \mathcal{R} \times \mathbb{R}^n \rightarrow [0, \infty)$ is continuously differentiable.

The solution to this problem has numerous applications that range from mobile robotics to path planning on topographical maps. In Section 3.1.5, we consider the specific application of computing paths for groups of Unmanned Aerial Vehicles (UAVs) that minimize the risk of being destroyed by ground defenses, *e.g.* Surface-to-Air Missiles (SAMs).

The computation of shortest paths has a long history and is one of the fundamental problems in Calculus of Variations (*cf.*, *e.g.*, [10]). Assuming for simplicity that $\mathcal{R} := \mathbb{R}^n$, the optimization formulated above is equivalent to the optimal control problem of finding a terminal time $T \geq 0$ and a unit velocity control $v : [0, T] \rightarrow \mathcal{V}$, where $\mathcal{V} := \{v \in \mathbb{R}^n : \|v\| = 1\}$, that minimizes the cost

$$J := \int_0^T \ell(x(t), v(t)) dt,$$

subject to the dynamics $\dot{x} = v$ and initial and terminal conditions $x(0) = x_i$, $x(T) = x_f$. This problem can be solved by using the Hamilton-Jacobi-Bellman (HJB) equation. Assuming that there exists a continuously differentiable solution $V : \mathbb{R}^n \rightarrow \mathbb{R}$ to the HJB equation

$$0 = \min_{v \in \mathcal{V}} H(x, v, \nabla_x V(x)), \quad \forall x \in \mathbb{R}^n,$$

with boundary condition $V(x_f) = 0$, where the Hamiltonian H is defined by

$$H(x, v, p) := \ell(x, v) + \langle p, v \rangle, \quad \forall x, v, p \in \mathbb{R}^n,$$

the optimal control $v^* : [0, T^*] \rightarrow \mathcal{V}$ is given by

$$v^*(t) = \arg \min_{v \in \mathcal{V}} H(x^*(t), v, \nabla_x V(x^*(t))),$$

where x^* denotes the optimal trajectory defined by $\dot{x}^* = v^*$ and $x^*(0) = x_i$. However, this method generally fails when the HJB equation or a relaxation of it has no continuously differentiable solution. Even when an appropriate solution exists (perhaps only a viscosity solution), it is often computationally difficult to find.

We address this difficulty by solving a discrete version of the continuous problem, which provides an approximate solution that is not necessarily optimal but whose cost can be made arbitrarily close to the minimum. We start by partitioning the region \mathcal{R} into a set of disjoint convex subregions and sampling along the boundaries of these subregions to extract a finite set of points $\mathcal{X} \in \mathcal{R}$. From this set of samples, we then construct a graph whose vertices are the points in \mathcal{X} and

whose edges connect all pairs of vertices belonging to a common element of the partition. The cost of each edge e is given by $J[\rho_e]$, where ρ_e is the straight-line unit-velocity path connecting the endpoints of the edge e . The graph optimization problem can then be solved using standard shortest path algorithms. The key step in this method is constructing a graph such that the optimal cost of the discrete path is close to the optimal path for the continuous problem.

Although literature on the anisotropic shortest path problem is quite limited, some discretization-based approaches have been proposed before. Rowe *et al.* considered the problem of computing minimum-energy paths for a ground vehicle moving between two points on a hilly terrain, described by a polyhedral approximation [26]. Using a simple model for the energy-cost that takes into account the grade of the climb and is therefore velocity dependent, they provided an exact algorithm with worst-case computational complexity $O(n^n)$. Lanthier *et al.* later provided an approximate algorithm for the anisotropic shortest path problem with polynomial complexity [20]. Because of the “curse of dimensionality”, the successful application of these methods to nontrivial problems depends crucially on the algorithm used to sample the region \mathcal{R} . Our approach addresses precisely this issue. Inspired by the worst-case cost bounds given in Section 3.1.2, we propose an efficient algorithm in Section 3.1.4 to sample \mathcal{R} that results in a small cost penalty with a relatively sparse sampling of \mathcal{R} . The underlying idea is to sample \mathcal{R} so that the density of sample points is higher in regions where the optimal path is more likely to deviate from a straight-line.

3.1.2 Local Discretization

To overcome the difficulties that arise in solving the weighted anisotropic shortest path problem exactly, one may finely sample the region \mathcal{R} and force the path to consist of the concatenation of several straight line segments between sample points. For a finite number of sample points, this procedure converts the original continuous shortest-path problem into a shortest-path problem on a finite graph.

The discretization method considered in [18] was based on a dense sampling of the entire search region and required that the piecewise linear paths could contain line segments connecting any two points within a ball of a certain radius. This section presents a streamlining of this approach, which consists of generating a convex partition of the search region, and sampling the region along the boundaries of this partition. In Section 3.1.3, we will see that this method achieves a substantial reduction in computation time over the method in [18] with no increase in the cost penalty.

Consider a finite convex partition of the region \mathcal{R} , *i.e.* a family of closed convex sets \mathcal{R}_i , $i \in \{1, 2, \dots, K\}$ such that their union is exactly \mathcal{R} and their interiors do not intersect. Now let \mathcal{X} be a finite set of points lying on the boundaries of the regions \mathcal{R}_i . Note that since the regions are closed, points on their boundaries may belong to multiple regions. Under this formulation, the set of admissible paths will consist of straight line segments connecting points that belong to a common element of the partition. In particular, we define a graph $G := (V, E)$ in which the vertex set V contains a vertex v_x located at each point $x \in \mathcal{X}$, and the edge set is defined by $E := \left\{ (v_x, v_{x'}) \mid \exists i : x, x' \in \tilde{\mathcal{R}}_i \right\}$, where $\tilde{\mathcal{R}}_j$ is a slight

enlargement of \mathcal{R}_i that is needed in the proof of Lemma 2. A *path in G* from $v_{x_i} \in V$ to $v_{x_f} \in V$ is a sequence of vertices (where $x_0 := x_i$ and $x_N := x_f$)

$$p := (v_{x_0}, v_{x_1}, \dots, v_{x_N}), \quad (v_{x_k}, v_{x_{k+1}}) \in E$$

for each $k \in \{1, \dots, N-1\}$. Let ρ_p denote the continuous path generated by connecting consecutive vertices v_{x_k} and $v_{x_{k+1}}$ in the path p by straight line segments. We denote the set of all possible paths ρ_p by \mathcal{P}_G .

The following Lemma proves the existence of a convex partition and set of sample points having properties that will allow us to construct a worst-case bound on the cost of a path in \mathcal{P}_G .

Lemma 1 *Given any positive constants $A, \epsilon_x, \epsilon_v, \epsilon_\delta$, there exists a finite convex partition $\{\mathcal{R}_1, \dots, \mathcal{R}_K\}$ of \mathcal{R} and a finite set \mathcal{X} of points in the boundaries of the regions \mathcal{R}_i such that for every twice continuously differentiable path $\rho : [0, T] \rightarrow \mathcal{R} \in \mathcal{P}$ with second derivative bounded by A , one can find*

1. a sequence of times $\{\tau_0, \tau_1, \dots, \tau_N\} \subset [0, T]$, with $\tau_0 := 0 \leq \tau_{k-1} < \tau_k \leq \tau_N := T$, and
2. a sequence of points $\{x_0 := \rho(0), x_1, \dots, x_N := \rho(T)\} \subset \bar{\mathcal{X}}$, where $\bar{\mathcal{X}} := \mathcal{X} \cup \{\rho(0), \rho(T)\}$,

such that

$$\|x_k - x_{k-1}\| \leq \epsilon_\delta, \quad \forall k \in \{1, \dots, N\}, \quad (3.1)$$

$$\|\rho(t) - x_{k-1}\| \leq \epsilon_\delta, \quad \forall t \in [\tau_{k-1}, \tau_k], \quad k \in \{1, \dots, N\}, \quad (3.2)$$

$$\|\rho(\tau_k) - x_k\| \leq \epsilon_x, \quad \forall k \in \{0, 1, \dots, N\}, \quad (3.3)$$

$$\left\| \dot{\rho}(\tau) - \frac{x_k - x_{k-1}}{\|\tau - \tau_{k-1}\|} \right\| \leq \epsilon_v, \quad \forall \tau \in (\tau_{k-1}, \tau_k), \quad k \in \{1, \dots, N\}, \quad (3.4)$$

$$\sum_{k=1}^N \|x_k - x_{k-1}\| \leq \frac{1 + \epsilon_v}{2 + \epsilon_v} 2T. \quad (3.5)$$

Moreover, as ϵ_x decreases to zero, the integer N remains uniformly bounded.

Without loss of generality, we assume that $\epsilon_\delta > \epsilon_x$.

See Appendix A for proofs of Lemma 1 and the following Lemma, which provides a bound on the difference in costs between two path segments.

Lemma 2 *Given two paths $\rho_i : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} , an interval $(t_1, t_2) \subset [0, T]$ on which each ρ_i is twice continuously differentiable and lies entirely inside the region $\tilde{\mathcal{R}}_j \subset \mathcal{R}$, and a constant $\delta > 0$,*

$$\int_{t_1}^{t_2} \ell(\rho_2(t), \dot{\rho}_2(t)) dt - \int_{t_1}^{t_2} \ell(\rho_1(t), \dot{\rho}_1(t)) dt \leq g_x \|\rho_2(t_1) - \rho_1(t_1)\| \Delta + g_v \|\dot{\rho}_2(t_1^+) - \dot{\rho}_1(t_1^+)\| \Delta + (g_x + \frac{a_1 + a_2}{2} g_v) \Delta^2 \quad (3.6)$$

where

$$g_x := \sup_{x \in \mathcal{R}_j, v \in \mathcal{V}} \|\nabla_x \ell(x, v)\|, \quad \Delta := \min \{ \|\rho_1(t_2) - \rho_1(t_1)\|, \|\rho_2(t_2) - \rho_2(t_1)\| \},$$

$$g_v := \sup_{x \in \mathcal{R}_j, v \in \mathcal{V}} \|\nabla_v \ell(x, v)\|, \quad a_i := \sup_{t \in (t_1, t_2)} \|\ddot{\rho}_i(t)\|,$$

$$\tilde{\mathcal{R}}_j := \mathcal{R}_j \cup \bigcup_{r \in \mathcal{R}_j} B_\delta(r).$$

We denote by \mathcal{V} the set of all unit velocities in \mathbb{R}^n , and $B_\delta(r)$ denotes the closed ball of radius δ centered at the point r .

Theorem 1 states the main result, *i.e.* that one can use the Honeycomb sampling method to construct a path that is arbitrarily close to the minimum-cost continuous path.

Theorem 1 *For every pair of constants $A, \epsilon > 0$, there exists a finite convex partition \mathcal{R}_i , $i \in \{1, 2, \dots, K\}$, of \mathcal{R} and a finite set \mathcal{X} of points in the boundaries of the regions \mathcal{R}_i such that for every initial and final points $x_i, x_f \in \mathcal{R}$*

$$\inf_{\rho \in \mathcal{P}_{\tilde{\mathcal{X}}}} J[\rho] \leq \inf_{\rho \in \mathcal{P}_{\|\ddot{\rho}\| \leq A}} J[\rho] + \epsilon,$$

where $\mathcal{P}_{\|\ddot{\rho}\| \leq A}$ denotes the set of twice continuously differentiable paths in \mathcal{P} with second derivative bounded by A .

Proof: Let $\rho^* : [0, T^*] \rightarrow \mathcal{R}$ be a path in $\mathcal{P}_{\|\ddot{\rho}\| \leq A}$ for which

$$J[\rho^*] \leq \inf_{\rho \in \mathcal{P}_{\|\ddot{\rho}\| \leq A}} J[\rho] + \delta,$$

where T^* is the time at which $\rho(t) = x_f$, and δ is a constant that can be made arbitrarily small (δ is only needed when the infimum is not a minimum).

We define \mathcal{R}_i , $i \in \{1, 2, \dots, K\}$ to be the finite convex partition of \mathcal{R} , and \mathcal{X} the finite set of points in the boundaries of the regions \mathcal{R}_i whose existence is guaranteed by Lemma 1. Now consider the sequences $\{\tau_0, \tau_1, \dots, \tau_N\} \subset [0, T^*]$ and $\{x_0, x_1, \dots, x_N\} \subset \mathcal{X}$ corresponding to the path ρ^* , which are also guaranteed to exist by Lemma 1. Suppose that we use the points x_k to construct the path $p := (v_{x_0}, v_{x_1}, \dots, v_{x_N})$ in the graph G . Let ρ_p denote the corresponding continuous path belonging to the set $\mathcal{P}_{\bar{\mathcal{X}}}$.

We proceed to compare the costs associated with ρ^* and ρ_p . To this effect, we expand

$$J[\rho_p] - J[\rho^*] = \sum_{k=1}^N \int_{\tau_{k-1}}^{\tau_k} \ell(\rho_p(s), \dot{\rho}_p(s)) ds - \int_{\tau_{k-1}}^{\tau_k} \ell(\rho^*(s), \dot{\rho}^*(s)) ds$$

Applying Lemma 2 to each interval (τ_{k-1}, τ_k) , $\forall k \in \{1, \dots, N\}$, we conclude that

$$\begin{aligned} J[\rho_p] - J[\rho^*] \leq \sum_{k=1}^N g_{x,k} \|\rho_p(\tau_k) - \rho^*(\tau_k)\| \Delta_k + g_{v,k} \|\dot{\rho}_p(\tau_k^+) - \dot{\rho}^*(\tau_k^+)\| \Delta_k \\ + \left(g_{x,k} + \frac{a_1 + a_2}{2} g_{v,k} \right) \Delta_k^2, \end{aligned} \quad (3.7)$$

where $g_{x,k}$, $g_{v,k}$, and Δ_k are as defined in Lemma 2 on each interval (τ_{k-1}, τ_k) for the paths ρ_p and ρ^* . Using the bounds provided by Lemma 1, we obtain

$$J[\rho_p] - J[\rho^*] \leq \sum_{k=1}^N \left((g_{x,k} \epsilon_x + g_{v,k} \epsilon_v + (g_{x,k} + A g_{v,k}) \Delta_k) \Delta_k \right) \leq \frac{2 + 2\epsilon_v}{2 + \epsilon_v} gT, \quad (3.8)$$

where

$$g := \sup_k (g_{x,k}\epsilon_x + g_{v,k}\epsilon_v + (g_{x,k} + Ag_{v,k})\epsilon_\delta).$$

This finishes the proof, since the right-hand-side of (3.8) can be made arbitrarily small by selecting ϵ_x , ϵ_δ , and ϵ_v sufficiently small. \square

3.1.3 Computational Complexity

In this section, we compare the computational complexity of the shortest-path approximation using the Honeycomb discretization method presented here, with that of the discretization method used in [18]. It turns out that there is a significant computational advantage gained by sampling on the boundaries of a convex partition rather than sampling the entire region. Since the expressions for the worst-case approximation bounds are roughly the same in both methods, we will compare the computation required by each method to obtain a path whose cost is within ϵ of the minimum path cost $J[\rho^*]$.

Consider a two-dimensional region $\mathcal{R} \subset \mathbb{R}^2$. For any positive constants A , ϵ_x , ϵ_v , and ϵ_δ , the method in [18] requires a density of nodes inversely proportional to ϵ_x^2 to guarantee the cost bound of $J[\rho^*] + \epsilon$. That is,

$$n = O\left(\frac{\mathcal{A}_{\mathcal{R}}}{\epsilon_x^2}\right),$$

where $\mathcal{A}_{\mathcal{R}}$ denotes the area of the region \mathcal{R} . Furthermore, each vertex in this graph must be connected to every vertex lying inside a ball of radius ϵ_δ by an

edge in the graph. The resulting number of edges is inversely proportional to ϵ_x^4 :

$$m = O\left(n \cdot \frac{\pi \epsilon_\delta^2}{\mathcal{A}_R} n\right) = O\left(\frac{\mathcal{A}_R \pi \epsilon_\delta^2}{\epsilon_x^4}\right).$$

In the Honeycomb sampling method, the number vertices required is proportional to the total length of the boundaries of the convex partition, which we approximate by assuming each element in the partition is a circle with diameter ϵ_δ . This leads to the following expression for the number of vertices:

$$n = O\left(\frac{\pi \epsilon_\delta^2}{\epsilon_x} \cdot \frac{\mathcal{A}_R}{\pi \left(\frac{\epsilon_\delta}{2}\right)^2}\right) = O\left(\frac{4\mathcal{A}_R}{\epsilon_x \epsilon_\delta}\right).$$

Each of these vertices must be connected to every other vertex belonging to the same partition element or within a short distance of this region (See Lemma 2). This results in a number of edges proportional to the number of total vertices multiplied by the number of vertices on the boundary of each element of the partition:

$$m = O\left(n \cdot \frac{\pi \epsilon_\delta}{\epsilon_x}\right) = O\left(\frac{\mathcal{A}_R \pi \epsilon_\delta^2}{\epsilon_x^4}\right).$$

We recall from Chapter 2 that the best-known computation for solving the all-pairs shortest-path problem is $O(nm + n^2 \log n)$ [17]. Substituting the values for m and n required by the discretization method in [18] yields the following expression for the computational complexity of that method:

$$O\left(\frac{\mathcal{A}_R^2 \pi \epsilon_\delta}{\epsilon_x^6} + \frac{\mathcal{A}_R^2}{\epsilon_x^4} \log \left[\frac{\mathcal{A}_R}{\epsilon_x^2}\right]\right)$$

For the Honeycomb discretization method, we have

$$O\left(\frac{4\pi\mathcal{A}_{\mathcal{R}}^2}{\epsilon_x^3\epsilon_\delta} + \frac{4\mathcal{A}_{\mathcal{R}}^2}{\epsilon_x^2\epsilon_\delta^2} \log\left[2\frac{\mathcal{A}_{\mathcal{R}}}{\epsilon_x\epsilon_\delta}\right]\right) \quad (3.9)$$

We see that as ϵ_x approaches zero, the $\frac{1}{\epsilon_\delta}$ term grows much faster than any other term. Therefore the Honeycomb discretization achieves a significantly lower computational complexity for generating an ϵ -optimal solution to the continuous weighted anisotropic shortest path problem. The following section describes a method for implementing a non-uniform sampling of the search region that takes advantage of the structure of the cost weighting function to reduce the total number of graph vertices and thus further reduce computation.

3.1.4 Non-uniform Sampling

In principle, since we have reduced the continuous weighted anisotropic shortest path optimization to a shortest path problem on a graph, the solution simply involves applying standard shortest path algorithms. The main difficulty with this approach is that the required computation time, given in (3.9), may grow very fast as desired accuracy of approximation (ϵ) decreases. However, we can minimize this effect by carefully selecting the location and relative size of the elements $\mathcal{R}_1, \dots, \mathcal{R}_K$ of the convex partition.

Theorem 1 proves the existence of a convex partition and a sampling on the boundaries of that partition on which one can construct a piecewise linear path ρ_p whose cost is within ϵ of the minimum path cost. Although no method is provided for generating the samples, Lemma 1 proves the existence of such a sample set \mathcal{X} that satisfies the following constraints for any path ρ^* :

1. the distance between consecutive sample points x_k should not exceed ϵ_δ ;
2. each sample point x_k should be in an ϵ_x -ball of the corresponding point $\rho^*(\tau_k)$ in the path;
3. the difference between the derivatives of ρ^* and its approximation ρ_p should not exceed ϵ_v .

By forcing ϵ_δ , ϵ_x , and ϵ_v to be sufficiently small, we can obtain an ϵ -optimal path ρ_p . However, a closer examination of Theorem 1 reveals that a careful selection of the partition and sample points may yield substantial computational benefit with minimal loss of optimality. Specifically, we observe that for the cost bound

$$J[\rho_p] - J[\rho^*] \leq \sum_{k=1}^N \left((g_{x,k}\epsilon_x + g_{v,k}\epsilon_v + (g_{x,k} + Ag_{v,k})\Delta_k) \Delta_k \right)$$

to be small, it suffices that ϵ_x , ϵ_v , and

$$g_{x,k}\epsilon_x + g_{v,k}\epsilon_v + (g_{x,k} + Ag_{v,k})\|x_k - x_{k-1}\| \tag{3.10}$$

be small for every k (recalling that $\|x_k - x_{k-1}\| \leq \Delta_k$). From this expression, we conclude that $\|x_k - x_{k-1}\|$ may actually be large in some regions, provided that $g_{x,k} + Ag_{v,k}$ is sufficiently small in those regions. This means that a large value may be assigned to ϵ_δ , allowing for large diameters of the partition elements \mathcal{R}_i , subject to the constraints already imposed on these diameters in the proof of Theorem 1. Indeed, by keeping the diameters of these regions inversely proportional to $g_{x,k} + g_{v,k}A$, we can significantly reduce the number of graph vertices and edges while still providing an ϵ -optimal path.

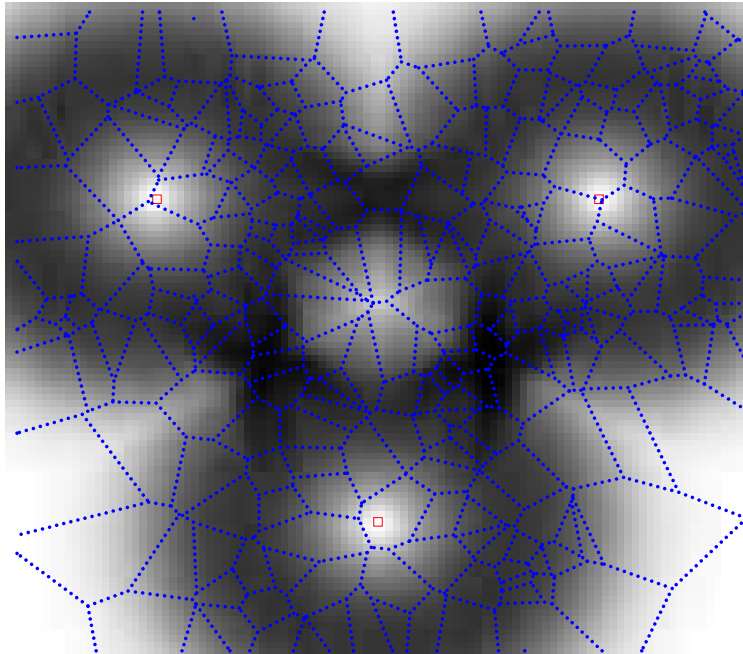


Figure 3.1. Honeycomb sampling

We now present a sampling algorithm for generating the sample points in this manner, which is based on the one introduced in [18]. However, we have now proved that this algorithm can be used to generate ϵ -optimal paths because the Voronoi diagram of step 2 can be used to generate the partition elements $\mathcal{R}_1, \dots, \mathcal{R}_K$ of Theorem 1, and step 3 generates the corresponding sample set \mathcal{X} on the boundaries of the regions \mathcal{R}_i .

Algorithm 1 (Honeycomb sampling)

1. Randomly choose K points $\mathcal{Z} := \{z_k \in \mathcal{R} : k = 1, 2, \dots, K\} \subset \mathbb{R}^n$, with a

spatial probability density over \mathcal{R} proportional to

$$\left(\sup_{v \in \mathcal{V}} \|\nabla_x \ell(x, v)\| + A \|\nabla_v \ell(x, v)\| \right)^n.$$

The distance between points in a particular area is then roughly inversely proportional to

$$K^{\frac{1}{n}} \sup_{v \in \mathcal{V}} \|\nabla_x \ell(x, v)\| + A \|\nabla_v \ell(x, v)\|. \quad (3.11)$$

If the diameters of any of these elements are greater than the limits imposed in Theorem 1, one may need to repeat this step with a larger value of K .

2. Compute the Voronoi diagram generated by the points in \mathcal{Z} . The size of the resulting cells in a particular area is roughly proportional to the distance between the points in that area, which is inversely proportional to (3.11).
3. Construct \mathcal{X} by sampling the edges of the Voronoi diagram sufficiently finely so that it is possible to choose points x_k in an ϵ_x -ball of any point where an optimal path would cross the cell boundary (See Lemma 1). □

Figure 3.1 shows an example of a set of sample points generated by this algorithm. In this figure, the background color represents the magnitude of $\sup_{v \in \mathcal{V}} \|\nabla_x \ell(x, v)\| + A \|\nabla_v \ell(x, v)\|$, with dark areas representing regions with large values. Increasing the number of Voronoi elements K , decreases the term $(g_{x,k} + A g_{v,k}) \|x_k - x_{k-1}\|$ in (3.10), whereas increasing the density of sampling over the edges of the Voronoi diagram decreases ϵ_x .

As in [18], we compare the Honeycomb sampling algorithm to several alternative sampling algorithms. For each of these methods, the graph is constructed by connecting each vertex v to all other vertices v' lying inside a ball of radius ϵ_δ of v . This allows us to guarantee the bounds constructed in [18]. The following two methods are the simplest and do not use information about the the structure of the cost weighting function ℓ :

Algorithm 2 (Regular-grid sampling) *Construct \mathcal{X} by overlaying a cubic lattice grid on \mathcal{R} with spacing equal to $\frac{2\epsilon_x}{\sqrt{3}}$ (to ensure the proper spacing).* \square

Algorithm 3 (Randomized uniform sampling) *Construct \mathcal{X} by randomly extracting N points with a uniform spatial probability density over \mathcal{R} .* \square

The following method attempts to minimize the term $(g_{x,k} + Ag_{v,k})\|x_k - x_{k-1}\|$ in (3.10).

Algorithm 4 (Randomized gradient-based sampling) *Construct \mathcal{X} by randomly extracting N points, with a spatial probability density over \mathcal{R} proportional to $(\sup_{v \in \mathcal{V}} \|\nabla_x \ell(x, v)\| + A\|\nabla_v \ell(x, v)\|)^n$.* \square

In the next section we compare the performance of these three sampling algorithms with the Honeycomb sampling algorithm in the context of minimum-risk path planning.

3.1.5 Minimum-Risk Path Planning for Groups of UAVs

In this section, we consider the cooperative routing mission described in [18] that involves transporting a group of M Unmanned Aerial Vehicles (UAVs) be-

tween two points in a hostile region $\mathcal{R} \subset \mathbb{R}^3$. The threats in this region are posed by k Surface-to-Air Missile (SAM) sites equipped with radar sensors located at positions $z_1, z_2, \dots, z_k \in \mathbb{R}^3$. Our goal is to compute the path between these points that maximizes the probability that the UAVs will survive the mission. In general, different UAVs may have different stealth capabilities and therefore their probabilities of survival are distinct. Because of this, minimum-risk path planning is a multi-criteria optimization problem. We will seek paths that are Pareto-optimal, *i.e.*, paths for which the probability of any single UAV surviving cannot be improved without decreasing the survivability of another UAV in the group.

If we denote the probability that the j^{th} UAV safely reaches the destination by $p_j^{\text{survive}}[\rho]$, we can obtain Pareto-optimal paths by solving single-criteria optimization problems of the form

$$\max_{\rho} \sum_{j=1}^m \lambda_j p_j^{\text{survive}}[\rho],$$

where the λ_j 's are positive constants [3]. Assuming that velocities are normalized so the maximum speed of the slowest UAV is equal to one, the optimization should be performed as ρ ranges over the set \mathcal{P} considered in the previous sections. Note that in general, risk is minimized for the maximum speed so there is no reason to consider paths with speeds smaller than the maximum. For the graph optimization, we ignore all constraints posed by the aircraft dynamics other than its maximum speed. However, in Section 3.1.5, we revisit these constraints and show how to construct feasible UAV paths from the piecewise linear paths resulting from the graph optimization.

Risk Model

Let us assume that the probability of the j th UAV being hit by the i th SAM in an elementary time interval dt is given by

$$\eta_{ij}(x, \dot{x}, z_i)dt,$$

where x and \dot{x} denote the position and velocity of the group of UAVs. The function η_{ij} is called the *risk density for the j th UAV with respect to the i th SAM*. In [18], it is shown that the probability of survival for a UAV can be expressed as

$$p_j^{\text{survive}}[\rho] = e^{-\int_0^T \ell_j(\rho(t), \dot{\rho}(t))dt},$$

where

$$\ell_j(x, v) := \sum_{i=1}^n \eta_{ij}(x, v, z_i). \quad (3.12)$$

This model is consistent with the expectation that if one remains under danger for a long amount of time, the probability of survival eventually converges to zero. Since the function $s \mapsto e^{-s}$ is monotone decreasing, paths that are Pareto-optimal (maximal) with respect to the rewards $p_j^{\text{survive}}[\rho]$ are also Pareto-optimal (minimal) with respect to the costs

$$J_j[\rho] := \int_0^T \ell_j(\rho(t), \dot{\rho}(t))dt. \quad (3.13)$$

We can therefore find these paths by solving the Weighted Anisotropic Shortest-Path Problem considered in Sections 3.1.2–3.1.4.

Simulations

We now simulate a scenario representing a typical minimum-risk path planning problem, which is illustrated in Figure 3.2. The small circles indicate a set

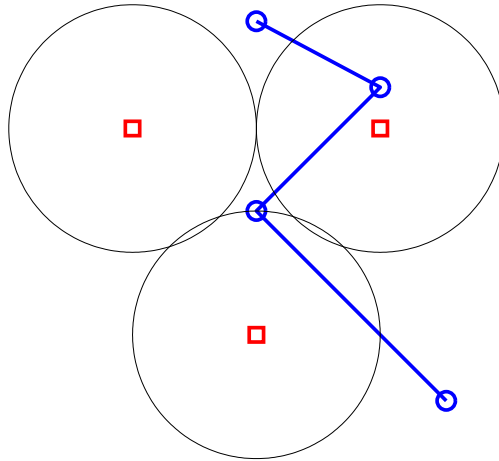


Figure 3.2. Minimum-risk path planning scenario.

of waypoints that the UAVs must visit starting at the top and finishing at the bottom right, but the straight line path drawn in the figure will generally not be the best path to take. On this mission, there is a threat posed by three SAM sites with radar sensors, which are represented by squares and are surrounded by large circles indicating their maximum effective ranges. The objective of this mission is to visit each of the waypoints while minimizing the risk of destruction by the SAMs.

For the risk density function, we use a model given by

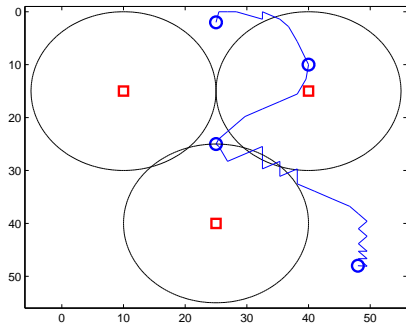
$$\eta_{ij}(x_j, \dot{x}_j, z_i) = \frac{1}{1 + \exp[h(x_j, \dot{x}_j, z_i)]}, \quad (3.14)$$

where

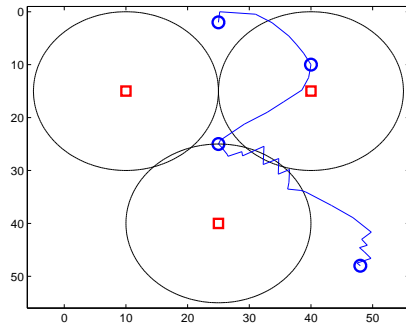
$$h(x_i, \dot{x}_j, z_i) = k [\|r_{ij}\| - ((r_2 - r_1)(1 - (r_{ij}^T v_j)^2) + r_i^1)].$$

In the expression above, r_{ij} is the radial vector from the i th SAM to the j th UAV, and v_j is the velocity vector of the j th UAV. The radii r_1 and r_2 determine the effective range of an SAM for a UAV flying directly toward it, and orthogonal to it, respectively. In between these flight vectors, there is a continuous dropoff in cost that begin roughly at a distance r_1 when the UAV is flying directly toward the SAM site, and at r_2 when the UAV is flying orthogonal to it. The constant k determines the steepness of this dropoff. Although (3.14) may be a useful model in practice, we use it primarily as an illustrative model for the demonstration of anisotropic shortest path algorithms, and we make no claims as to the accuracy of this model for real UAVs.

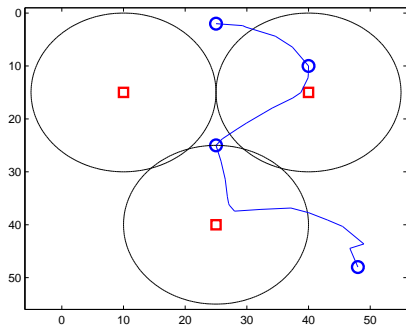
Figure 3.3 shows the top-view of the paths computed by each method for this scenario. The altitude computation was decoupled by first computing the minimum path cost at the maximum altitude, and then sampling the resulting path at various altitudes and computing the minimum-cost path on that set. Figure 3.4 shows a three-dimensional view of the path computed by the Honeycomb sampling method. The path takes advantage of the altitude control within the specified range $(0.1, 1)$, by heading down toward the SAM sites and then up away from them.



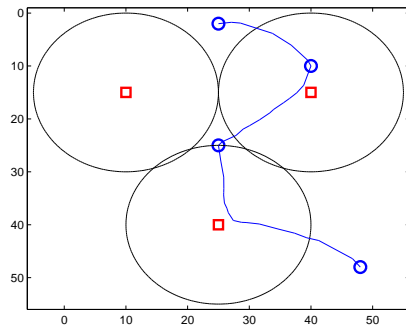
(a) Grid sampling



(b) Uniform random sampling



(c) Gradient-based random sampling



(d) Honeycomb sampling

Figure 3.3. Paths computed for this scenario using various sampling algorithms.

Figure 3.3 shows the paths computed by each method for this scenario, and Table 3.1 lists the corresponding costs and computation times. The minimum path cost computed using each method is approximately the same, but as expected, the Honeycomb sampling method facilitates a much faster computation due to the lower number of edges in the graph. Both the regular grid and uniform random sampling methods are able to find paths that achieve slightly lower cost by making a series of sharp turns between the outer ranges of two of the SAM sites. Depending on the dynamical constraints of the UAV, these may or may not be feasible paths. In Section 3.1.5, we provide an cubic-spline smoothing algorithm designed for these situations. At the cost of some extra computation,

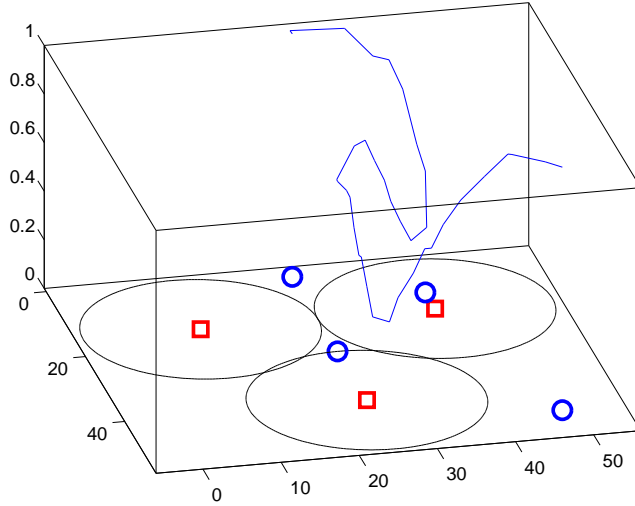


Figure 3.4. Three dimensional view of path.

Table 3.1. Comparison of Sampling Algorithms for Minimum-Risk Path Planning

Sampling Method	Cost	Vertices	Edges	Comp. Time (s)
Regular grid	27.0	3490	221000	12.5
Uniform random	27.0	4450	372000	21.7
Gradient-based random	27.4	4450	511000	29.7
Honeycomb	27.3	3820	92400	5.3

this algorithm not only smooths the paths to meet velocity and acceleration constraints, but in many cases also lowers the path costs.

Figure 3.5 shows a plot of computation time vs. minimum path cost for various values of ϵ_x and ϵ_δ . In general, the Honeycomb sampling method results in some of the lowest costs with consistently low computation times compared to the other methods. Table 3.2 contains the means and standard deviations of this data.

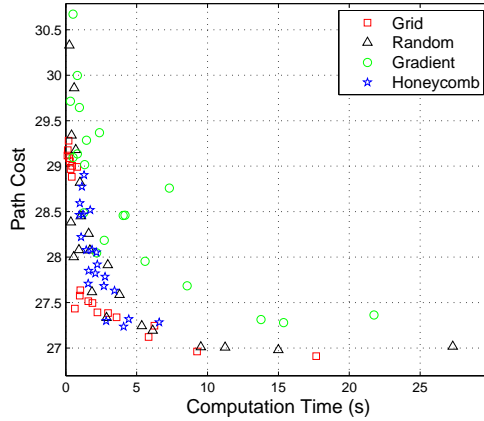


Figure 3.5. Path cost vs. computation time.

Table 3.2. Means and Standard Deviations of the Path Costs

Sampling Method	Mean	Standard Deviation
Regular grid	28.1	0.90
Uniform random	28.1	0.97
Gradient-based random	28.6	1.0
Honeycomb	28.0	0.50

Spline Smoothing Optimization

One aspect of the routing problem that was not taken into account in the previous section is constraints on the path imposed by dynamics of the UAV. When constructing a continuous path out of piecewise linear segments, one must assume that the UAV can turn fast enough to render any transitive effects between segments negligible with respect to the path-cost. Although this may be a valid assumption in some instances, it is not generally the case. We address this issue here by proposing a post-processing algorithm that combines gradient-descent optimization of the waypoints with a cubic spline interpolation of the path. Although this method requires some additional computation, the resulting

path is guaranteed to meet velocity and acceleration constraints of a UAV, and in many cases, its cost is as good or better than that of the piecewise linear path. The algorithm is described in detail below.

We start with the path $p := (v_{\text{init}}, v_2, \dots, v_{\text{final}})$ obtained from the graph optimization with the sampling set of Theorem 1. To keep the cubic spline curve from deviating too far from this path, we insert additional waypoints at time intervals T_s wherever path vertices are separated by a time greater than T_s . Let w denote a vector of waypoints. We will require the cubic spline path to pass through these waypoints, also referred to as *knots*. However, for the optimization, we will relax the position constraints of all but the initial and final points in the path.

The final spline curve will consist of $N := |w| - 1$ cubic segments. Between two knots w_i and w_{i+1} occurring at times t_i and t_{i+1} , we can express the path as

$$y_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3,$$

for $i \in \{1, 2, \dots, N\}$, where a_i, b_i, c_i, d_i are vector coefficients of the i^{th} spline segment. The coefficients of a cubic spline must satisfy

$$a_i = w_i, \quad \forall i \in \{1, \dots, N\}, \quad (3.15)$$

$$a_i + b_i\Delta_i + 2\Delta_i^2c_i + 3\Delta_i^3d_i = a_{i+1}, \quad \forall i \in \{1, \dots, N-1\}, \quad (3.16)$$

$$b_i + 2\Delta_ic_i + 3\Delta_i^2d_i = b_{i+1}, \quad \forall i \in \{1, \dots, N-1\}, \quad (3.17)$$

$$2c_i + 6\Delta_ic_i = c_{i+1}, \quad \forall i \in \{1, \dots, N-1\}, \quad (3.18)$$

where $\Delta_i := t_{i+1} - t_i$. Condition (3.15) requires each segment to start at its

corresponding waypoint. The remaining conditions require adjacent segments to be continuous (3.16), continuously differentiable (3.17), and twice continuously differentiable (3.18). We add a final condition

$$a_N + b_N \Delta_N + 2\Delta_N^2 c_N + 3\Delta_N^3 d_N = w_{N+1}, \quad (3.19)$$

which requires the last segment to finish at the final waypoint. Typically, one would now add zero-acceleration constraints at the endpoints to generate a full system of equations. For our purposes, this is unnecessary because we do not seek a unique solution.

Since our objective here is to further improve the path, not just smooth it, we relax condition (3.15) on the interior knots by restating it as

$$\|a_i - w_i\| \leq \delta \quad \forall i \in \{2, \dots, N\}, \quad (3.20)$$

for some constant $\delta > 0$.

Next, we add velocity and acceleration constraints relating to dynamic limitations of the UAVs, *i.e.*

$$\|b_i\| \leq V_{\max}, \quad \forall i \in \{1, \dots, N\}, \quad (3.21)$$

$$\|b_N + 2\Delta_N c_N + 3\Delta_N^2 d_N\| \leq V_{\max}, \quad (3.22)$$

$$\|c_i\| \leq A_{\max}, \quad \forall i \in \{1, \dots, N\}, \quad (3.23)$$

$$\|c_N + 6\Delta_N d_N\| \leq A_{\max}, \quad (3.24)$$

where V_{\max} and A_{\max} are the maximum velocity and acceleration of the UAVs.

As mentioned previously, risk is minimized for high velocities, so although the UAVs will also have a minimum velocity, it is not necessary to enforce this as a constraint. Note that if A_{\max} is chosen to be equal to the acceleration bound A from Section 3.1.2, we will obtain a path that belongs to the class $\mathcal{P}_{\|\rho\|\leq A}$ of twice continuously differentiable paths with second derivative bounded by A .

With these constraints defined, we are now ready to formulate our post-processing optimization problem. Let $S := [A, B, C, D]$ be the concatenation of the spline coefficients, *i.e.* $A := (a_1, \dots, a_{|w|-1})$ and similarly for B, C, D . We can express our problem as follows:

Spline Smoothing Optimization: Given a path ρ_p generated by the discrete minimum-risk path-planning algorithm, compute the minimum-cost path

$$\rho_S^*(t) := \arg \min_{\rho \in \mathcal{P}_S} \int_0^T \ell(\rho(t), \dot{\rho}(t)) dt,$$

where \mathcal{P}_S is the class of cubic splines, determined by the coefficients S , that satisfy constraints (3.16)-(3.24). Figure 3.6 shows the result of this cubic spline smoothing procedure performed using MATLAB's *fmincon* nonlinear optimization algorithm. The velocity and acceleration constraints are $V_{\max} = 1$ and $A_{\max} = 1$, and the maximum position deviation is $\delta = 3$. Figure 3.7 shows a three-dimensional view of this path.

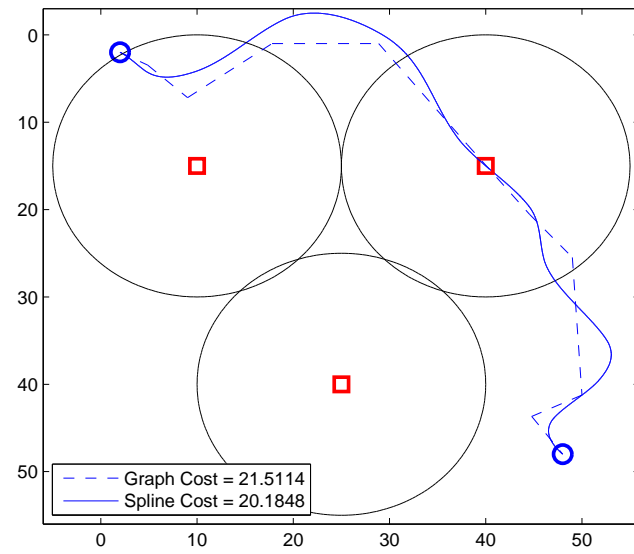


Figure 3.6. Cubic-spline path smoothing

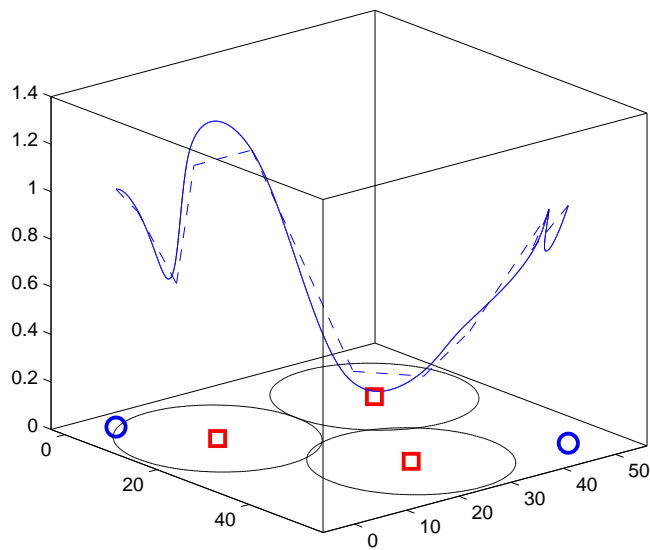


Figure 3.7. Three-dimensional view of smoothed path

Figure 4.1 is a diagram of the fractal decomposition process on a graph with a clustered structure. On the first decomposition level, the 420-vertex graph is partitioned into 20 subgraphs, each containing 21 vertices. The upper arrow points to the 20 vertex *meta-graph*, and the lower arrow points to the 20 subgraphs. These 21 new graphs are further decomposed into a total of 85 even smaller graphs. This illustrates the main idea behind the fractal decomposition method, that we can reduce the computation required to solve a large complex graph optimization problem by decomposing it into smaller problems of the same form. The solutions to these smaller problems are then used to generate an approximate solution on the original graph. We constructed the graph in Figure 4.1 specifically to illustrate the recursive decomposition process, but the method works on any graph. Increasing the number of decomposition levels greatly reduces computation, but generally results in looser bounds. These bounds depend on the specific instance of the problem, but we show that problem-independent approximation bounds do exist for certain regular graphs such as lattices, which could appear to be poorly suited to hierarchical decomposition due to their uniform structure.

This chapter begins with a literature review on hierarchical decomposition of graph problems in Section 4.1. In Section 4.2, we present the method of fractal decomposition in general terms, followed by computational complexity results in Section 4.3. Sections 4.4, 4.5, and 4.6 provide detailed examples of the fractal decomposition procedure performed on shortest path matrix, maximum flow matrix, and cooperative graph search problems, respectively. Each of these sections also contains a numerical simulation as well an analysis on the tightness of the approximation bounds for lattice graphs.

4.1 Related work

There is some previous literature on hierarchical decomposition applied to various graph optimization problems, most prevalently the shortest path problem. Romeijn and Smith proposed an algorithm to approximately solve the all-pairs shortest path problem using hierarchical decomposition. Under the assumption that graphs in each level of aggregation have the same structure, they showed the computational complexity of their approximation (using parallel processors) to be $O(n \log n)$ for aggregation on two levels of sparse graphs, and $O(n^{\frac{2}{L}} \log n)$ for aggregation on L levels. They also provide an upper bound for this approximation.

A notable application of the shortest path problem is dynamic programming for optimal control. One can think of optimal control as the problem of finding the minimum cost path over all sequences of control inputs that transfer a system from an initial to a desired state, and dynamic programming is a well-known technique for solving this problem. Shen and Caines presented results on “Hierarchically Accelerated Dynamic Programming” [27] with applications to optimal control. Using state aggregation methods, they were able to speed up dynamic programming algorithms for finite state machines by orders of magnitude at the expense of some sub-optimality, for which they give upper bounds. Our approach to the fractal shortest path decomposition example will closely resemble that of [25] and [27], but with the addition of lower bounds on the costs of the shortest paths. This gives the user the advantage of being able to gauge how far the resulting approximation is from the true minimum cost path.

We are more interested, however, in problems with higher computational com-

plexity, because it is in these problems that there is the most to be gained by a hierarchical decomposition. One such problem is maximum flow. Towards approximating the maximum flow problem, Lim *et al.* developed a technique to compute routing tables for stochastic network routing that involves a two-level hierarchical decomposition of the network. They reduced computational complexity from $O(n^5)$ to $O(n^{3.1})$ with a performance that is in some cases as good as the general flat max-flow routing problem [21]. Our maximum flow decomposition improves the two-level computational complexity to $O(n^3)$ and adds the capability to decompose on more levels using the fractal framework.

The final problem we discuss is the search problem, which is the most computationally complex of the problems considered here, and is the primary focus of this dissertation. There is very limited literature related to the hierarchical decomposition of the search problem. The inspiration for our fractal decomposition approach to the cooperative search problem was a paper by DasGupta *et al.*, which presents an approximate solution to the continuous “Honey-pot” constrained search problem based on a discrete aggregation of the search space [5]. After approximating the solution to the discrete search problem with a two-level hierarchical decomposition, they refine the resulting discrete path into a continuous one, providing bounds on the suboptimality resulting from discretization as well as the approximation of the discrete problem. Here, we expand upon these ideas to allow for implementation on multiple hierarchical levels and to include cooperative search problems involving multiple agents.

4.2 Method

We now describe a methodology for bounding and approximating the solution to a graph optimization problem using fractal decomposition. We will apply the ideas presented here to several example problems in the following sections. The three key steps of the fractal decomposition method are:

1. Partition the graph
2. Construct bounding meta-problems and solve
3. Refine worst-case solution to approximate solution on original graph

Step 2 is where the recursion occurs because the resulting meta-problems are in the same form as the original problem, and thus may also be solved using the fractal decomposition method. We now summarize each of the key steps in the algorithm.

4.2.1 Partition the graph

Although the fractal decomposition algorithm will work for any graph partition, the specific partition used will have significant effects on the computation as well as the accuracy of the resulting approximation. The choice of partitioning algorithm also depends on the problem under consideration. In general, the first objective of the graph partition is to decompose the problem such that the difference between upper and lower bounds is small. For example, in the shortest path problem, this means grouping vertices that are connected by low-cost edges whereas in the maximum flow problem, this means grouping vertices connected

by high-capacity edges. See Section 2.2.1 for a more detailed discussion of graph partitioning.

4.2.2 Construct bounding meta-problems and solve

The next step is to construct two *meta-problems* using the results of the graph partition: a *worst-case* meta-problem and a *best-case* meta-problem. Both meta-problems share the same meta-graph defined by the partition, and they should be formulated in the exact same way as the original problem, so that we may apply any algorithm that solves the original problem to the meta-problems. The most important step in this construction is assigning data (costs, rewards, capacities, *etc.*) to the meta-vertices and meta-edges such that the solutions to the meta-problems are guaranteed to bound the optimal value of the original problem. For the worst-case meta-problem, this step involves assigning conservative values to the meta-edges and meta-vertices to ensure that its optimal value will be achievable in the original problem. The values for each meta-vertex generally come from solving a smaller problem, defined on the corresponding subgraph. For the best case meta-problem, one assigns optimistic values so that its solution will be better than optimal on the original graph, although not achievable. Once constructed and solved, the solution to the worst-case meta-problem will yield the conservative bound and will facilitate the generation of an approximate solution on the original graph. The solution to the best-case meta-problem provides a measure of how far our approximate solution is from optimal.

4.2.3 Refine worst-case solution to approximate solution on original graph

As mentioned above, the solution to the worst-case meta-problem will involve solving a set of smaller subproblems. One can use the solutions to these smaller problems to generate an approximate solution to the original problem. This is made possible because the worst-case meta-problem is formulated in such a way that the resulting approximate solution is guaranteed to be feasible on the original graph.

4.3 Computational Complexity

Let us now look at how much the fractal decomposition algorithm can reduce computational complexity. For the purposes of this analysis, let $f(n, k)$ denote the computational complexity of a given graph optimization problem on n vertices decomposed into k meta-vertices. Without decomposition, the complexity is $f(n, 1)$. Now, let us see what happens with one level of decomposition. Suppose that we partition the graph into k subgraphs each containing roughly $\frac{n}{k}$ vertices. The computational complexity of the worst-case decomposed problem is then

$$f(n, k) = f(k, 1) + kf\left(\frac{n}{k}, 1\right), \quad (4.1)$$

where the first term comes from solving a problem on the meta-graph, and the second term comes from solving problems on the k subgraphs. Decomposing on

a second level yields

$$\begin{aligned}
f(n, k_0) &= f(k_0, k_1) + k_0 f\left(\frac{n}{k_0}, k_2\right) \\
&= f(k_1, 1) + k_1 f\left(\frac{k_0}{k_1}, 1\right) \\
&\quad + k_0 f(k_2, 1) + k_0 k_2 f\left(\frac{n}{k_0 k_2}, 1\right). \tag{4.2}
\end{aligned}$$

A choice of $k = \sqrt{n}$ in (4.1) makes the computation of the upper-level meta-problem equal to that of the k subproblems. For two levels, the analogous choices are $k_0 = \sqrt{n}$, and $k_1 = k_2 = \sqrt{k_0} = \sqrt[4]{n}$. Continuing the recursive decomposition in this manner yields the following computational complexity results:

$$\begin{aligned}
\text{Level 0 : } & f(n, 1) \\
\text{Level 1 : } & (\sqrt{n} + 1)f(\sqrt{n}, 1) \\
\text{Level 2 : } & (\sqrt{n} + 1)(\sqrt[4]{n} + 1)f(\sqrt[4]{n}, 1) \\
& \vdots \\
\text{Level L : } & \sum_{i=0}^{2^L-1} n^{\frac{i}{2^L}} f(n^{\frac{1}{2^L}}, 1) \\
& = \left(\frac{n-1}{n^{\frac{1}{2^L}}-1} \right) f(n^{\frac{1}{2^L}}, 1).
\end{aligned}$$

We can limit the number of decomposition levels to $L_{\max} = \lceil \log_2(\log_2(n)) \rceil$ because there is no advantage to decomposing a graph with only two vertices. It follows that as L approaches this maximum value, the computational complexity approaches $(n-1)f(2, 1)$, which is equivalent to $O(n)$ since $f(2, 1)$ is a constant. Hence, as the number of decomposition levels increases, the complexity

approaches linearity.

Inherent to this analysis is the assumption that the computational complexity of a given problem can be expressed solely as a function of the number of vertices in the graph. This is not generally the case because the complexity of many algorithms also depends on the number of edges m . However, we can obtain an upper bound on the computational complexity by analyzing the results for dense graphs, where $O(m) = O(n^2)$. Similarly, setting $O(m) = O(n)$ yields the best-case complexity for sparse graphs. Table 4.1 shows the computational reduction for various levels of decomposition on the three example problems presented in the following sections.

Table 4.1. Computational Complexity of Worst-Case Meta-problems for up to 3 Decomposition Levels on Dense Graphs

Levels	Shortest Path Matrix	Maximum Flow Matrix	Cooperative Search
0	$O(n^3)$	$O(n^5)$	$O(n^M!)$
1	$O(n^2)$	$O(n^3)$	$O(n^{\frac{1}{2}}(n^{\frac{M}{2}})!)$
2	$O(n^{\frac{3}{2}})$	$O(n^2)$	$O(n^{\frac{3}{4}}(n^{\frac{M}{4}})!)$
3	$O(n^{\frac{5}{4}})$	$O(n^{\frac{3}{2}})$	$O(n^{\frac{7}{8}}(n^{\frac{M}{8}})!)$

In the cooperative search column, M is the number of searchers, and the complexities listed are for an exhaustive search. For this analysis, we assume that the complexity associated with graph partitioning is negligible compared to that of the optimization problem, which may or may not be the case depending on the specific partitioning algorithm used.

4.4 Shortest Path

In this section, we show how to apply the fractal decomposition technique to the shortest path matrix problem. Although the shortest path problem is computationally the least complex of the examples we consider, the procedure for constructing the worst-case and best-case meta-graphs is quite illustrative and probably the most intuitive.

The shortest path matrix problem involves constructing a $n \times n$ matrix of the minimum-cost path between all pairs of vertices in a graph. Our formulation is a slight variation on the conventional all-pairs shortest paths problem because in addition to assigning a cost to each edge, we also assign a cost to each vertex. This is crucial in facilitating the hierarchical decomposition of the problem, as will be explained in the construction of the worst-case meta-problem. Adding vertex costs does not increase the computational complexity of the problem.

4.4.1 Problem Formulation

Data

$G := (V, E)$	directed graph
$c_e : E \rightarrow [0, \infty)$	edge cost function
$c_v : V \times O \rightarrow [0, \infty)$	vertex cost function

Path A *path* in G from $v_{\text{init}} \in V$ to $v_{\text{final}} \in V$ is a sequence of vertices (where

$v_1 := v_{\text{init}}, v_\ell := v_{\text{final}}$)

$$p := (v_1, v_2, \dots, v_{\ell-1}, v_\ell), \quad (v_i, v_{i+1}) \in E.$$

The *path-cost* is given by

$$C(p) := \sum_{i=1}^{\ell-1} c_e(v_i, v_{i+1}) + \sum_{i=1}^{\ell} c_v(v_i).$$

Objective For every pair of vertices $(v_{\text{init}}, v_{\text{final}})$, compute the path p that minimizes the path-cost $C(p)$. We denote this path by p^* and the minimum cost $C(p^*)$ by $J_G^*(v_{\text{init}}, v_{\text{final}})$. The largest minimum cost over all possible pairs of vertices $(v_{\text{init}}, v_{\text{final}})$ is called the *diameter* of the graph, which we denote by $\|G\|_D$.

Worst-case meta-shortest path Let $\bar{G}_{\text{worst}} := (\bar{V}, \bar{E})$ be a meta-graph of G , with edge cost and vertex cost defined by

$$\bar{c}_{e_{\text{worst}}}(\bar{e}) := \min_{e \in \bar{e}} c_e(e) \quad \bar{c}_{v_{\text{worst}}}(\bar{v}) := \|G|_{\bar{v}}\|_D.$$

Note that to assign the vertex costs of \bar{G}_{worst} one needs to compute the diameter of all subgraphs and therefore solve k smaller shortest-path matrix problems, where k is the number of meta-vertices. This is the key step in the hierarchical decomposition of the problem.

Best-case meta-shortest path Let $\bar{G}_{\text{best}} := (\bar{V}, \bar{E})$ be a meta-graph of G ,

with edge and vertex costs defined by

$$\bar{c}_{e_{\text{best}}}(\bar{e}) := \min_{e \in \bar{e}} c_e(e) \qquad \bar{c}_{v_{\text{best}}}(\bar{v}) := \min_{v \in \bar{v}} c_v(v).$$

Theorem 2 *For every partition \bar{V} of G and for every pair of vertices v_{init} and v_{final} ,*

$$J_{\bar{G}_{\text{best}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \leq J_G^*(v_{\text{init}}, v_{\text{final}}) \leq J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \quad (4.3)$$

where $v_{\text{init}} \in \bar{v}_{\text{init}}$, $v_{\text{final}} \in \bar{v}_{\text{final}}$. Additionally, the graph diameters satisfy

$$\|\bar{G}_{\text{best}}\|_D \leq \|G\|_D \leq \|\bar{G}_{\text{worst}}\|_D. \quad (4.4)$$

In what follows, the construction of the upper bound provides a procedure for generating an approximate shortest path between each pair of vertices in G , and the cost of this path lies between $J_G^*(v_{\text{init}}, v_{\text{final}})$ and $J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}})$.

Proof: To verify the upper bound, we will show that one can use the minimum-cost path from \bar{v}_{init} to \bar{v}_{final} in \bar{G}_{worst} to construct an approximate minimum-cost path from v_{init} to v_{final} in G . We do this by sequentially connecting v_{init} , the endpoints of the minimum cost edge between each meta-vertex in the optimal worst-case path, and v_{final} with the minimum-cost path between them in G . The procedure is described below.

Let \bar{p}_w^* be the shortest path from \bar{v}_{init} to \bar{v}_{final} in \bar{G}_{worst} , and \tilde{p} the approximate shortest path in G being constructed. To begin, we force \tilde{p} to include the

minimum cost edge of each meta-edge in \bar{p}_w^* . We denote this set of edges by E_w^* . From the set of vertices adjacent to these edges, let $v_{j_{\text{exit}}}$ be the exit vertex of \bar{v}_j , that is, the vertex in \bar{v}_j adjacent to the edge in E_{worst}^* associated with the meta-edge that connects \bar{v}_j to \bar{v}_{j+1} for $j = (1, 2, \dots, \bar{\ell} - 1)$, and let $v_{j_{\text{entry}}}$ be the entry vertex of each \bar{v}_j for $j = (2, 3, \dots, \bar{\ell})$. The incomplete path is then

$$\begin{aligned} \tilde{p} = & (v_{\text{init}}, \dots, v_{\text{exit}_1}, \dots, v_{\text{entry}_2}, \dots, v_{\text{exit}_2}, \dots \\ & \dots, v_{\text{exit}_{\bar{\ell}-1}}, \dots, v_{\text{entry}_{\bar{\ell}}}, \dots, v_{\text{final}}). \end{aligned}$$

Now, simply fill in the gaps with the minimum-cost path between the surrounding vertices. Recall that we already computed the shortest paths between all pairs of vertices in a meta-vertex when we found the diameter of each subgraph $G|\bar{v}_j$, so this data is available without any additional computation.

Let \tilde{p}_j denote the portion of the path \tilde{p} contained within the meta-vertex \bar{v}_j . We can now compare the costs of the two paths \tilde{p} and \bar{p}_w^* :

$$C(\bar{p}_w^*) = \sum_{j=1}^{\bar{\ell}-1} \bar{c}_{e_{\text{worst}}}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{\ell}} \|G|\bar{v}_j\|_D, \quad (4.5)$$

$$C(\tilde{p}) = \sum_{j=1}^{\bar{\ell}-1} \bar{c}_{e_{\text{worst}}}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{\ell}} C(\tilde{p}_j), \quad (4.6)$$

where $\tilde{\ell}$ is the number of vertices in the path \tilde{p} . Because we have defined each \tilde{p}_j to be the shortest path between two nodes in \bar{v}_j , the cost of this path $C(\tilde{p}_j)$ must be no greater than $\|G|\bar{v}_j\|_D$. Summing over the same set of meta-vertices, the last term of (4.6) must be less than or equal to the last term of (4.5). Since

the edge cost terms are identical, we conclude that

$$C(p^*) \leq C(\tilde{p}) \leq C(\bar{p}_w^*), \quad (4.7)$$

where p^* is the shortest path in G from v_{init} to v_{final} . The left inequality in (4.7) holds because of the optimality of p^* , and the right inequality was already discussed. Therefore, the upper bound in (5) holds. Since the upper bound holds for every pair of vertices in G , we know that the diameter of \bar{G}_{worst} must bound the diameter of G from above, that is, $\|G\|_D \leq \|\bar{G}_{\text{worst}}\|_D$.

We now verify the lower bound by constructing a feasible path from \bar{v}_{init} to \bar{v}_{final} in \bar{G}_{best} from the optimal path p^* connecting v_{init} to v_{final} in G . The constructed path, which we will call \bar{p}_b , will consist of the sequence of meta-vertices that contain vertices of p^* in order but with no consecutive repetitions. We can construct this path by first setting $\bar{p}_b = p^*$ and then replacing each vertex v_i with the meta-vertex \bar{v}_j in which it is contained. Deleting all repeated meta-vertices yields the desired path $\bar{p}_b = (\bar{v}_1, \bar{v}_2, \dots, \bar{v}_{\bar{\ell}})$, where $\bar{\ell}$ is the length of \bar{p}_b .

Let E_{p^*} denote the set of edges along the path p^* . Define $\hat{E}_{p_j^*}$ as the set of these edges having both endpoints in \bar{v}_j , and \hat{E}_{p^*} as the set of all edges totally contained within meta-vertices, for $j \in [1, \bar{\ell}]$. The remaining edges connecting consecutive \bar{v}_j along p^* are contained in the set difference $\tilde{E}_{p^*} := E_{p^*} \setminus \hat{E}_{p^*}$.

We can express the cost of the two paths as follows:

$$C(\bar{p}_{\text{best}}) = \sum_{j=1}^{\bar{\ell}-1} \bar{c}_{e_{\text{best}}}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{\ell}} \bar{c}_{v_{\text{best}}}(\bar{v}_j) \quad (4.8)$$

$$C(p^*) = \sum_{e \in \hat{E}_{p^*}} c_e(e) + \sum_{e \in \hat{E}_{p^*}} c_e(e) + \sum_{j=1}^{\ell} c_v(v_j), \quad (4.9)$$

where ℓ is the length of p^* .

The first term on the right side of (4.9) is the cost of the edges in p^* between meta-vertices and this is greater than or equal to the sum of the minimum edge costs between the same sequence of meta-vertices, which is the first term on the right side of (4.8). Using this and the trivial fact that the sum of the minimum vertex costs in each meta-vertex, which is the second term on the right of (4.8), is less than or equal to the sum of total costs incurred by p^* within meta-vertices, the second and third terms on the right of (4.9), we see that the lower bound in (5) indeed holds. Since the lower bound holds for every pair of vertices in G , it is also true that $\|\bar{G}_{\text{best}}\|_D \leq \|G\|_D$. \square

4.4.2 Approximation Error Bounds for Lattice Graphs

The approximation bounds discussed thus far are problem dependent, that is, varying the graph, graph data, or partition will also vary the resulting bounds. A natural question to ask is whether it is possible to guarantee bounds that do not depend on the particular instance of the problem. This is quite a difficult task for arbitrary graphs, but it turns out that we can indeed generate constant factor approximation bounds for the diameters of certain regular graphs, such

as lattices. Though lattices are a simple class of graphs, they would seem to be a particularly bad case for hierarchical decomposition algorithms because there is no clustering of vertices and therefore no structurally advantageous way to partition them.

Let $Z_d(n)$ denote a d -dimensional lattice graph with n vertices. To begin, let us consider the shortest-path problem on two-dimensional square lattice graphs $Z_2(n)$ with dimensions $\sqrt{n} \times \sqrt{n}$. To allow for a uniform graph partition and to simplify the results, we restrict the lattice sizes to $n = i^2 \times i^2$, for $i = 2, 3, \dots$, but we expect that the results will hold for all other sizes with an appropriate irregular partition.

The diameter of $Z_2(n)$ is $2(\sqrt{n} - 1)$. We now apply one level of decomposition to the problem, dividing the graph into \sqrt{n} blocks each having \sqrt{n} vertices. The diameter of the resulting worst-case meta-graph is $4(n^{\frac{1}{2}} - n^{\frac{1}{4}})$. The ratio of these diameters is

$$\frac{\|\bar{G}_{\text{worst}}^1\|_D}{\|Z_2(n)\|_D} = \frac{4(n^{\frac{1}{2}} - n^{\frac{1}{4}})}{2(n^{\frac{1}{2}} - 1)} = \frac{2n^{\frac{1}{4}}}{n^{\frac{1}{4}} + 1} \leq 2 \quad \forall n \in \mathbb{N} \setminus \{1\}.$$

If we again use the fractal decomposition algorithm to approximate the subgraph diameters, which generate the meta-vertex costs of \bar{G}_{worst}^1 , we incur an additional error factor that is no greater than 2. Recursively applying this result gives a constant approximation factor of 4 for two decomposition levels, 8 for three, and 2^L for L . For cubic lattice graphs, the approximation factor turns out to be 3^L . We now generalize this result to d -dimensional *hypercubic* lattice graphs, *i.e.* lattice graphs in d dimensions with an equal size in each dimension.

Lemma 3 For d -dimensional hypercubic lattice graphs of size $n = (i^2)^d$, $\forall i \in \mathbb{N} \setminus \{1\}$,

$$\|\bar{G}_{\text{worst}}^L\|_D \leq d^L \|Z_d(n)\|_D,$$

where L is the number decomposition levels used in the fractal algorithm.

Proof: The diameter of $Z_d(n)$ is $d(n^{\frac{1}{d}} - 1)$. Partitioning the graph into $\sqrt[n]{n}$ hypercubic lattice graphs each having $\sqrt[n]{n}$ vertices results in a worst-case meta-graph diameter of $d^2 n^{\frac{1}{d}} + (2d - 2d^2)n^{\frac{1}{2d}} + (d^2 - 2d)$. The ratio of these diameters is

$$\frac{\|\bar{G}_{\text{worst}}^L\|_D}{\|Z_d(n)\|_D} = \frac{d^2 n^{\frac{1}{d}} + (2d - 2d^2)n^{\frac{1}{2d}} + (d^2 - 2d)}{d(n^{\frac{1}{d}} - 1)} \leq d \quad \forall n \in \mathbb{N} \setminus \{1\}.$$

For each additional level of decomposition applied, we incur an additional error factor that is no greater than d . This results in a constant approximation factor of d^2 for two decomposition levels and d^L for L levels. \square

4.4.3 Case Study on Shortest Path Routing

This section presents an application of the fractal decomposition algorithm to a multiple-agent shortest-path routing problem. Suppose there are M autonomous mobile agents at various locations in a region with many obstacles. Each agent has a destination somewhere else in the region and their goal is to arrive there in minimum time. First, we generate a set of graph nodes from the region by sampling based on a gradient of the obstacle map, shown in Fig-

ure 4.2(a). We also include nodes for the agents' locations and destinations. Then we construct the graph edges using a Delaunay triangulation (See Figure 4.2(b)).

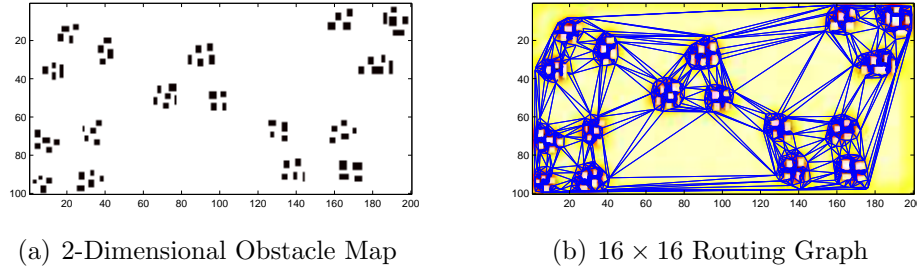


Figure 4.2. The left figure shows an overhead view of obstacles in a region. The graph to the right is generated from sampling where the gradient of the obstacle map is high, and connecting nodes with a Delaunay triangulation.

The diameter is an appropriate metric for the tightness of the shortest path bounds, because for each hierarchical level below the top, subgraph diameters are computed to assign the meta-vertex costs. Table 4.2 shows the results for best-case, worst-case, and actual diameters for the routing graph. The worst-case diameter bound is within about 10% of the actual diameter, and the best-case bound is within 52%. The computation times for the meta-problems include the graph partitioning process and are two orders of magnitude faster than that of the original problem.

Table 4.2. Diameter Approximation

	Best-case	Actual	Worst-case
Diameter	114.2	236.8	262.8
Computation Time	10.4	1250	10.4

We also generated approximate shortest paths for 100 agents starting at random nodes and traveling to random destination nodes. The approximate minimum path cost will always lie between the actual minimum path cost and the worst-case upper bound. Table 4.3 shows the results. Clearly there will be zero

error when the source and destination nodes are contained in the same meta-vertex. The mean error for this set of trials was 8.1%.

Table 4.3. Shortest Path Approximation for 100 Agents

Minimum % Error	0
Maximum % Error	24
Mean % Error	8.1

We can conclude that the fractal decomposition approximation generates quite good results to the shortest path matrix problem for graphs with some inherent clustered structure. For graphs without such structure, the approximation may be significantly less accurate, but in S 4.4.2, we showed that for lattice graphs, which have no clustering, we can approximate the diameter to within a constant factor of d^L .

4.5 Maximum Flow

In the maximum flow matrix problem, the goal is to construct a matrix containing the maximum flow intensities between all pairs of vertices in a graph. The flow through each edge is limited by the capacity of that edge. In this formulation, the flow through each vertex is also limited by a capacity. This vertex capacity is what allows for the hierarchical decomposition of this problem.

4.5.1 Problem Formulation

Data

$G := (V, E)$	directed graph
$c_e : E \rightarrow [0, \infty)$	edge capacity function
$c_v : V \times O \rightarrow [0, \infty)$	vertex capacity function

Flow A *flow in G* from $v_{\text{init}} \in V$ to $v_{\text{final}} \in V$ is a function $f : E \rightarrow [0, \infty)$ for which there exists some $\mu \geq 0$ such that

$$f_{\text{out}}(v) - f_{\text{in}}(v) = \begin{cases} \mu, & v = v_{\text{init}} \\ -\mu, & v = v_{\text{final}} \\ 0, & \text{otherwise,} \end{cases} \quad \forall v \in V, \quad (4.10)$$

$$0 \leq f(e) \leq c_e(e), \quad \forall e \in E \quad (4.11)$$

$$0 \leq f_{\text{in}}(v) \leq c_v(v), \quad \forall v \in V, \quad (4.12)$$

$$0 \leq f_{\text{out}}(v) \leq c_v(v), \quad \forall v \in V. \quad (4.13)$$

In the above,

$$f_{\text{in}}(v) := \sum_{e \in \text{In}[v]} f(e), \quad f_{\text{out}}(v) := \sum_{e \in \text{Out}[v]} f(e), \quad (4.14)$$

where $\text{In}[v] \in E$ denotes the set of edges that are directed toward the vertex v and $\text{Out}[v] \in E$ denotes the set of edges that are directed out from vertex v . The

constant μ is called the *intensity* of the flow.

Objective For every pair of vertexes $(v_{\text{init}}, v_{\text{final}})$, compute the flow f^* with maximum intensity μ from v_{init} to v_{final} . The maximum intensity is denoted by $J_G^*(v_{\text{init}}, v_{\text{final}})$ and is called the *maximum flow from v_{init} to v_{final}* . The smallest maximum flow over all possible pairs of vertices is called the *capacity* of the graph and is denoted by $\|G\|_C$.

Worst-case meta-max flow Let $\bar{G}_{\text{worst}} := (\bar{V}, \bar{E})$ be a meta-graph of G , with edge capacity and vertex capacity defined by

$$\bar{c}_{e_{\text{worst}}}(\bar{e}) := \sum_{e \in \bar{e}} c_e(e) \qquad \bar{c}_{v_{\text{worst}}}(\bar{v}) := \|G|\bar{v}\|_C. \quad (4.15)$$

Note that to construct the graph \bar{G}_{worst} one needs to compute the capacity of all subgraphs and therefore solve several smaller max-flow matrix problems.

Best-case meta-max flow Let $\bar{G}_{\text{best}} := (\bar{V}, \bar{E})$ be a meta-graph of G , with edge capacity and vertex capacity defined by

$$\bar{c}_{e_{\text{best}}}(\bar{e}) := \sum_{e \in \bar{e}} c_e(e) \qquad \bar{c}_{v_{\text{best}}}(\bar{v}) := \sum_{v \in \bar{v}} c_v(v). \quad (4.16)$$

Theorem 3 For every partition \bar{V} of G and for every pair of vertices v_{init} and v_{final} ,

$$J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \leq J_G^*(v_{\text{init}}, v_{\text{final}}) \leq J_{\bar{G}_{\text{best}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \quad (4.17)$$

where $v_{\text{init}} \in \bar{v}_{\text{init}}$, $v_{\text{final}} \in \bar{v}_{\text{final}}$. Additionally, the graph capacities satisfy

$$\|\bar{G}_{\text{worst}}\|_C \leq \|G\|_C \leq \|\bar{G}_{\text{best}}\|_C. \quad (4.18)$$

In the proof of Theorem 2, the construction of the lower bound contains a procedure for generating an approximate maximum flow between each pair of vertices in G , and the intensity of this flow lies between $J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}})$ and $J_G^*(v_{\text{init}}, v_{\text{final}})$.

To verify the lower bound, the idea is to construct a flow $\tilde{f}(e)$ in G from the worst-case maximum flow $\bar{f}_w^*(\bar{e})$ in \bar{G}_{worst} , that satisfies (4.10)–(4.13). This is possible because the subgraphs associated with each meta-vertex are connected and the total flow into and out of a meta-vertex is always no greater than the capacity of that meta-vertex. The intensity of this approximate maximum flow is bounded below by $J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}})$, and above by $J_G^*(v_{\text{init}}, v_{\text{final}})$.

Proof: Let $\bar{f}_w^*(\bar{e})$ be the maximum flow assignment of \bar{G}_{worst} , from which we now construct a flow $\tilde{f}(e)$ in G that satisfies (4.10)–(4.13). For every $\bar{e} \in \bar{E}$, decompose the flow in the worst-case meta-graph as a flow in the original graph as follows:

$$\bar{f}_w^*(\bar{e}) = \tilde{f}(\bar{e}(1)) + \tilde{f}(\bar{e}(2)) + \cdots + \tilde{f}(\bar{e}(p)), \quad (4.19)$$

where the notation $\bar{e}(i)$ is used to index the edges associated with the meta-edge \bar{e} , and p is the total number of these edges. Since (4.11) holds for the worst-case

meta-graph, that is

$$0 \leq \bar{f}_w^*(\bar{e}) \leq \bar{c}_{e_{\text{worst}}}(\bar{e}) = \sum_{e \in \bar{e}} c_e(e),$$

we know that a decomposition (4.19) exists whose flows satisfy condition (4.11) for the original graph. Now we have assigned flows to the subset of edges of G that connect meta-vertices of \bar{G}_{worst} , but we still have to consider the edges of G that lie inside meta-vertices.

For every $\bar{v} \in \bar{V}$, there exist sets v'_{in} and v'_{out} defined by

$$v'_{in} = \{v \in \bar{v} : (u, v) \in E, u \notin \bar{v}, v \in \bar{v}\} \cup (\{v_{\text{init}}\} \cap \bar{v})$$

$$v'_{out} = \{v \in \bar{v} : (u, v) \in E, u \in \bar{v}, v \notin \bar{v}\} \cup (\{v_{\text{final}}\} \cap \bar{v})$$

The set v'_{in} consists of all vertices in \bar{v} to which an edge originating outside the meta-vertex is directed, plus the source vertex if $\bar{v} = \bar{v}_{\text{init}}$. Similarly, the set v'_{out} consists of all vertices in \bar{v} from which an edge directed outside the meta-vertex originates, plus the sink vertex if $\bar{v} = \bar{v}_{\text{final}}$. Some vertices may be contained in both v'_{in} and v'_{out} . To separate these vertices, we define the following disjoint sets:

$$v_{in} = \left\{ v \in v'_{in} : \tilde{f}_{in}(v) - \tilde{f}_{out}(v) > 0 \right\}$$

$$v_{out} = \left\{ v \in v'_{out} : \tilde{f}_{in}(v) - \tilde{f}_{out}(v) < 0 \right\},$$

where \tilde{f}_{in} and \tilde{f}_{out} are as defined in (4.14). Recall that we have only assigned $\tilde{f}(e)$ to edges connecting meta-vertices. For all other edges, $\tilde{f}(e)$ is temporarily set to zero.

We can now define a set of k subproblems, one for each $\bar{v} \in \bar{V}$, where the goal is to find a feasible flow from v_{in} to v_{out} . The following procedure will find such a flow. For simplicity, assume that flow originating at the source v_{init} is an inflow, and the flow terminating at the sink v_{final} is an outflow.

1. Enumerate the sets $v_{in} = \{v_{in_1}, v_{in_2}, \dots, v_{in_p}\}$ and $v_{out} = \{v_{out_1}, v_{out_2}, \dots, v_{out_q}\}$
2. Initially, $i = 1$ and $j = 1$.
3. If $G|\bar{v}$ is not connected, then the capacity of this meta-vertex \bar{v} is zero. Therefore, $\bar{f}_{out}^*(\bar{v}) = \bar{f}_{in}^*(\bar{v}) = 0$ and we are done. If $G|\bar{v}$ is connected, then there exists at least one path from v_{in_i} to v_{out_j} . Assign the flow along one such path to be the minimum of the flow into v_{in_i} and the flow out of v_{out_j} . Subtract this value from the flow intensities of both vertices.
4. If $i = p$ and $j = q$ then stop. The procedure is complete.
5. If the flow into v_{in_i} is still greater than the flow out, repeat step 3 for v_{in_i} and $v_{out_{j+1}}$. Otherwise execute step 3 for $v_{in_{i+1}}$ and v_{out_j} .

By construction, assigning flows in this manner preserves condition (4.10) because at the end of the procedure the net flow is zero at all vertices excluding v_{init} and v_{final} , for which it is $\tilde{\mu}$ and $-\tilde{\mu}$, respectively. Also, since the flow through a meta-vertex \bar{v} is bounded above by the capacity of the subgraph $G|\bar{v}$, the flows generated by this procedure will satisfy (4.11)-(4.13). This completes our construction of a feasible flow $\tilde{f}(e)$ in the original graph based on the worst-case flow $\bar{f}_w^*(\bar{e})$ in the meta-graph. Therefore,

$$J_G^*(v_{init}, v_{final}) \geq J_{\bar{G}_{worst}}^*(\bar{v}_{init}, \bar{v}_{final}). \quad (4.20)$$

Since the lower bound holds for every pair of vertices in G , we know that the capacity of \bar{G}_{worst} must bound the capacity of G from below, that is, $\|\bar{G}_{\text{worst}}\|_C \leq \|G\|_C$.

The proof for the best-case upper bound is straightforward because we can easily construct a flow in \bar{G}_{best} from an optimal flow $f^*(e)$ in G . Let $\hat{f}_{\text{best}}(\bar{e})$ be a not necessarily optimal flow in \bar{G}_{best} , where

$$\hat{f}_{\text{best}}(\bar{e}) = \sum_{e \in \bar{e}} f^*(e).$$

Since we know that $f^*(e)$ satisfies (4.10) and (4.11), it follows from (4.16) and the above equation that $\hat{f}_{\text{best}}(\bar{e})$ satisfies (4.10) and (4.11). Since $\bar{c}_{v_{\text{best}}}(\bar{v}) = \sum_{v \in \bar{v}} c_v(v)$, (4.12) and (4.13) must hold also. Let $\hat{J}_{\bar{G}_{\text{best}}}(\bar{v}_{\text{init}}, \bar{v}_{\text{final}})$ be the intensity of the flow $\hat{f}_{\text{best}}(\bar{e})$. As expected,

$$J_{\bar{G}_{\text{best}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \geq \hat{J}_{\bar{G}_{\text{best}}}(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) = J_G^*(v_{\text{init}}, v_{\text{final}}).$$

The equality on the right holds from our construction of $\hat{f}_{\text{best}}(\bar{e})$. The inequality on the left holds by definition of optimality. Since the upper bound holds for every pair of vertices in G , it is also true that $\|G\|_C \leq \|\bar{G}_{\text{best}}\|_C$. \square

4.5.2 Approximation Error Bounds for Lattice Graphs

We now address the issue of whether it is possible to guarantee a constant factor maximum flow approximation for regular lattice graphs. First, consider an undirected two-dimensional square lattice graph $G := L^{\sqrt{n} \times \sqrt{n}}$, in which all edge capacities are one and all vertex capacities are infinity. To allow for a

uniform graph partition and to simplify the results, we restrict the lattice sizes to $n = i^2 \times i^2$, for $i = 2, 3, \dots$, but we expect that the results will hold for all other sizes with an appropriate irregular partition.

In a two-dimensional lattice graph, vertices have degree two at the corners, three on outside edges between corners, and four on all interior edges. The maximum flow intensity between two vertices of degree four is 4 because there are four separate paths between them with no coinciding edges. The minimum max-flow occurs between two vertices of degree two, where the maximum flow intensity is 2 because there are only two independent paths between the vertices. Hence the capacity of every two-dimensional square lattice graph is 2. Partitioning the graph into \sqrt{n} square lattice graphs each having \sqrt{n} vertices results in a worst-case meta-graph with all meta-edge capacities equal to 2 and all meta-vertex capacities equal to 2. The capacity of this meta-graph is 2, the same as the capacity of the original graph. We now generalize this result to d -dimensional lattices.

Lemma 4 *For d -dimensional hypercubic lattice graphs of size $n = i^2d$, $\forall i \in \mathbb{N} \setminus \{1\}$,*

$$\|\bar{G}_{\text{worst}}(n)\|_C = \|G(n)\|_C.$$

Proof: This proof is similar to the two-dimensional case, but the minimum degree of a vertex in a d -dimensional lattice is d and the maximum degree is $2d$. Hence, the diameter is d because there are only d independent paths between pairs of minimum degree vertices. Partitioning the lattice into $n^{\frac{1}{2}}$ d -dimensional

hypercubes each having $n^{\frac{1}{2}}$ vertices results in a worst-case meta-graph with all meta-edge capacities equal to $n^{\frac{d-1}{2d}}$ and all meta-vertex capacities equal to d . The diameter of this meta-graph is d , the same as the diameter of the original graph. \square

The above results were made possible largely because the capacity of a finite lattice graph is determined by the flows to and from the corner vertices (minimum degree vertices), and corner vertices of the same degree exist in the original graph as well as the partitioned subgraphs. As a result, the maximum flow between corner vertices on the worst-case meta-graph is a feasible flow on the original graph.

We now consider the case in which the lattice wraps around itself such that the degree of all vertices is the same. This type of lattice is called a periodic or toroidal lattice graph.

Lemma 5 *For d -dimensional toroidal lattice graphs of size $n = i^2d$, $\forall i \in \mathbb{N} \setminus \{1\}$,*

$$\|\bar{G}_{\text{worst}}(n)\|_C = \frac{1}{2}\|G(n)\|_C.$$

Proof: The diameter of a d -dimensional toroidal lattice graph is $2d$ because there are $2d$ paths with no coinciding edges between every pair of vertices. Using the same partitions as for the non-toroidal lattice results in the same meta-graph as in the proof of Lemma 4 with the addition of the toroidal meta-edges. However, since the meta-vertex capacities do not change, the capacity of the meta-graph remains equal to d . \square

4.5.3 Case Study on Stochastic Max-Flow Routing

Suppose there are M autonomous mobile agents in a region with many obstacles, and each agent is trying to reach a destination in the same region. The difference from the shortest path routing problem in Section 4.4.3 is that we now have an adversary with the ability to intercept an agent. Suppose that the probability of interception is related to the length of a graph edge by $p_{int}(e) = 1 - \exp[-kc_e(e)]$, where k is a positive constant. In [2], Bohacek *et al.* showed that optimal routing policies in this game theoretic framework can be computed by solving a maximum flow problem on a graph whose edge capacities are given by $\frac{1}{p_{int}(e)}$. The resulting maximum flow assignment on the edges is used to determine which paths the agents should take with what probabilities to minimize the probability of being intercepted on their route. Hence, we use this assignment for the edges in our graph, having the same structure as the graph in Figure 4.2(b).

Table 4.4 shows the results of applying the fractal max-flow approximation to this routing problem with one level of decomposition. To partition the graph, we used the algorithm in [14], where the cost of cutting an edge e was given by $p_{int}(e)$. This causes the algorithm to group vertices connected by edges with low capacity so that the meta-graph will have relatively high edge capacities. The

Table 4.4. Capacity Approximation

	Best-case	Actual	Worst-case
Diameter	24	16	4
Computation Time	10.4	1250	10.4

worst-case capacity bound is within a factor of one-fourth of the true capacity and is computed in a time two orders of magnitude less than the computation

time for the full problem. The best-case capacity bound shows that in this case, an approximate routing policy implemented using the results of the worst-case optimization will perform no worse than one-sixth as well as the optimal routing policy.

4.6 Cooperative Graph Search

Let us now consider the problem in which a team of M agents is searching for one or more stationary targets in a bounded region represented by a graph. Each vertex has a reward, generally relating to the probability of finding an object at that vertex, and a cost representing a quantity such as time or energy spent searching that vertex. Each edge also has a cost, representing the cost incurred in transit between vertices. The team's goal is to find paths on the graph for each agent that maximize the total reward collected by the team subject to a cost constraint on the individual agents.

4.6.1 Problem Formulation

Data

$G := (V, E)$	directed location graph
$O := \{1, \dots, o_{\max}\}$	vertex occupancy set
$r : V \times O \rightarrow [0, \infty)$	vertex reward function
$c_v : V \times O \rightarrow [0, \infty)$	vertex cost function
$c_e : E \rightarrow [0, \infty)$	edge cost function
$L \in [0, \infty)$	cost bound

In the above, $o_{\max} \in \{1, 2, \dots, M\}$ is the maximum number of searchers allowed to occupy a single vertex. Note that the vertex cost and reward functions depend on the occupancy of the vertex. The reason for this will be made clear when we construct the meta-problems.

Search Path A *search path* in G is a sequence of vertices,

$$p := (v_1, v_2, \dots, v_{\ell-1}, v_{\ell}), \quad (v_i, v_{i+1}) \in E,$$

where ℓ is the length of the path. The *path-cost* is given by

$$C(p) := \sum_{i=1}^{\ell-1} c_e(v_i, v_{i+1}) + \sum_{i=1}^{\ell} c_v(v_i),$$

and the *path-reward* is given by

$$R(p) := \sum_{v \in p} r(v), \quad (4.21)$$

where the sum in (4.21) is taken with no repetitions, that is, if a vertex appears in p more than once, it is only included in the summation once. This represents the fact that the reward of a vertex can only be collected once.

The search problem for a single agent is to find the path p that maximizes the reward $R(p)$ subject to the cost constraint $C(p) \leq L$. We now extend the graph search problem to the case of multiple agents.

Cooperative Framework There is significant existing literature on the single-agent search problem, but the cooperative search problem is inherently more complex. One way to approach the multiple-agent problem would be to set up M identical single-agent search problems on the location graph G and have the agents start in different strategic positions, but this is not a cooperative solution and could result in overlapping search paths. For the team to fully cooperate, we must consider the problem as a whole. We can do this by creating a graph in which a vertex represents the locations of all agents, which implies that the full graph will consist of up to n^M nodes. We call this new expanded graph the *team-graph induced by G* and denote it by $\mathbf{G} := (\mathbf{V}, \mathbf{E})$ (Bold face notation is used for all data and functions related to the team-graph).

The *team-vertex* set \mathbf{V} consists of M -length vectors whose entries are the vertex locations in V of each member of the team. We write the expanded team-vertex as $\mathbf{v} = (\mathbf{v}(1), \mathbf{v}(2), \dots, \mathbf{v}(M))$, where $\mathbf{v}(a) \in V$ is the location of agent a when the team is at team-vertex \mathbf{v} . We construct the set \mathbf{V} by including

team-vertices for all possible configurations of agents in V such that the vertex occupancy o_{\max} is not exceeded. An edge connects vertices \mathbf{v} and \mathbf{v}' if there is an edge in E between $\mathbf{v}(a)$ and $\mathbf{v}'(a)$ for each agent a . We also allow members of the team to stay at a vertex. This is useful in the case that some agents reach their cost limit before others. We can write the team-edge set as

$$\mathbf{E} := \{(\mathbf{v}, \mathbf{v}') : \forall a (\mathbf{v}(a), \mathbf{v}'(a)) \in E \cup (\mathbf{v}(a), \mathbf{v}(a))\}, \quad \forall \mathbf{v}, \mathbf{v}' \in \mathbf{V}.$$

Team Search Path We now describe how to construct and evaluate paths in the team-graph based on data for the location graph. A *team search path* in \mathbf{G} is a sequence of vertices,

$$\mathbf{p} := (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{\ell-1}, \mathbf{v}_\ell), \quad (\mathbf{v}_i, \mathbf{v}_{i+1}) \in \mathbf{E}, \quad (4.22)$$

where ℓ is the length of the path. Let \mathbf{p}_a denote the path of agent a , that is $\mathbf{p}_a := (\mathbf{v}_1(a), \dots, \mathbf{v}_\ell(a))$. The *team path-cost* is the maximum path-cost of any agent in the team and is given by

$$\mathbf{C}(\mathbf{p}) := \max_a C(\mathbf{p}_a)$$

The *team path-reward* is the total reward collected by the search team on the path, which we can express as

$$\mathbf{R}(\mathbf{p}) := \sum_{i=1}^{\ell} \sum_{v \in \mathbf{v}_i} r(v, o_{\mathbf{v}_i}^v) \kappa(v, i), \quad (4.23)$$

where $o_{\mathbf{v}_i}^v$ is the number of agents occupying vertex v when the team vertex is \mathbf{v}_i .

The function

$$\kappa(v, i) := \begin{cases} 1, & v \notin \bigcup_{j=1}^{i-1} \mathbf{v}_j \\ 0, & \text{otherwise} \end{cases}.$$

encodes the property that the reward for a vertex in G may only be collected once by the search team.

Objective Given a cost bound L , let $J_G^*(M, L)$ denote the maximum reward that M searchers can collect on G . We express the objective for the cooperative search problem as follows:

$$J_G^*(M, L) := \max_{\mathbf{p}} \mathbf{R}(\mathbf{p}) \quad \text{s. t.} \quad \mathbf{C}(\mathbf{p}) \leq L. \quad (4.24)$$

Worst-case cooperative meta-graph search Let $\tilde{G}_{\text{worst}} := (\bar{V}, \bar{E})$ be a meta-graph of G . Our goal is to formulate the worst-case meta-problem such that its solution will be a lower bound on the optimal reward of (4.24).

We first choose a meta-vertex maximum occupancy \bar{o}_{\max} , yielding the occupancy set $\bar{O} := \{1, \dots, \bar{o}_{\max}\}$, and then choose a cost bound assignment $l : \bar{V} \times \bar{O} \rightarrow [0, \infty)$. These choices are a degree of freedom for the user, but they should be chosen carefully as they may significantly affect the tightness of the bounds on the optimal reward. The best choices will depend on the structure of the graph as well as the number of decomposition levels. One possible assignment is to simply choose $o_{\max} = 1$ and $l(\bar{v}, \bar{o}) = L$ for every meta-vertex \bar{v} .

The meta-vertex cost and reward functions are defined by solving cooperative

search problems on their corresponding subgraphs:

$$r_{\text{worst}}(\bar{v}, \bar{o}) := J_{G|\bar{v}}^*(\bar{o}, l), \quad (4.25)$$

$$c_{v_{\text{worst}}}(\bar{v}, \bar{o}) := \mathbf{C}(\mathbf{p}^*(\bar{v}, \bar{o})), \quad \forall \bar{o} \in \bar{O}, \forall \bar{v} \in \bar{V}, \quad (4.26)$$

where $\mathbf{p}^*(\bar{v}, \bar{o})$ is the optimal team search path in $G|\bar{v}$ that generates the maximum reward $J_{G|\bar{v}}^*(\bar{o}, l)$. Let $p_a^*(\bar{v}, \bar{o}, i)$ denote the i^{th} vertex in agent a 's optimal path on meta-vertex \bar{v} , having occupancy \bar{o} . We define the cost of an edge from \bar{v} to \bar{v}' by pairing all final vertices of paths computed in \bar{v} with all starting vertices of paths computed in \bar{v}' , computing the costs of the shortest paths between them, and taking the maximum of these costs. Figure 4.3 diagrams this process for two meta-vertices for which $o_{\max} = 2$. Suppose that the dotted lines represent optimal paths computed on both meta-vertices for an occupancy of 1, and the dashed lines are the paths computed for an occupancy of 2. The highlighted vertices represent the pair of ($\{\text{final vertices in } \bar{v}\}, \{\text{starting vertices in } \bar{v}'\}$) that are farthest apart. The cost of the shortest path between these vertices is 7, hence $c_{e_{\text{worst}}}(\bar{v}, \bar{v}') = 7$ in this example. We can formally write the worst case meta-edge

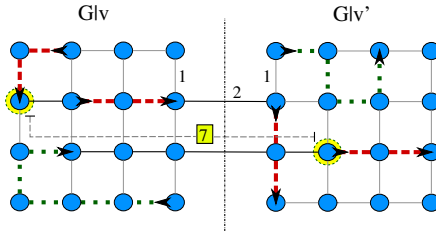


Figure 4.3. Example showing the worst-case edge-cost between two meta-vertices. Edge-costs are 1 for edges within subgraphs, 2 for edges between subgraphs and all vertex costs are 0.

cost function for all $(\bar{v}, \bar{v}') \in \bar{E}$, as

$$c_{e_{\text{worst}}}(\bar{v}, \bar{v}') = \max_{\bar{o}, \bar{o}'} \max_{a, a'} d[p_a^*(\bar{v}, \bar{o}, \ell), p_{a'}^*(\bar{v}', \bar{o}', 1)], \quad (4.27)$$

where $\bar{o}, \bar{o}' \in \bar{O}$, $a \in \{1, \dots, \bar{o}\}$, $a' \in \{1, \dots, \bar{o}'\}$, and $d[v, v']$ is the cost of the shortest path in G from v to v' . In implementation, there are ways to make this less conservative. For example, one could create a team edge-cost function that depends on the occupancies of adjacent vertices. However, for simplicity, we choose an edge-cost on the location meta-graph that does not depend on the vertex occupancies.

Now let $\bar{G}_{\text{worst}} := (\bar{V}, \bar{E})$ be the team meta-graph induced by \bar{G}_{worst} . The team path-cost and path-reward functions for the meta-graph are defined exactly the same as they were for the original graph in (4.22) and (4.23). The objective in the worst-case cooperative meta-graph search problem is to solve the cooperative search problem (4.24) on \bar{G}_{worst} and thus find the reward $J_{\bar{G}_{\text{worst}}}^*(M, L)$.

Best-case cooperative meta-graph search We construct the best-case problem such that its solution will be an upper bound on the optimal reward of (4.24). Let $\bar{G}_{\text{best}} := (\bar{V}, \bar{E})$ be a meta-graph of G with edge cost function defined by

$$c_{e_{\text{best}}}(\bar{v}, \bar{v}') = \min_{v \in \bar{v}, v' \in \bar{v}'} d[v, v'], \quad \forall (\bar{v}, \bar{v}') \in \bar{E}. \quad (4.28)$$

where $d[v, v']$ is the cost of the shortest path in G from v to v' (for the example in Figure 4.3, $c_{e_{\text{best}}}(\bar{v}, \bar{v}') = 2$). Now, set $o_{\text{max}} = 1$ and define the meta-vertex

reward and cost functions as

$$r_{\text{best}}(\bar{v}, 1) := \sum_{v \in \bar{v}} \max_o r(v, o) \quad (4.29)$$

$$c_{v_{\text{best}}}(\bar{v}, 1) := \min_{v \in \bar{v}} c_v(v, 1). \quad (4.30)$$

Let $\bar{\mathbf{G}}_{\text{best}} := (\bar{\mathbf{V}}, \bar{\mathbf{E}})$ be the team meta-graph constructed from \bar{G}_{best} . The objective in the best-case cooperative meta-graph search problem is to solve the cooperative search problem (4.24) on $\bar{\mathbf{G}}_{\text{best}}$ and thus find the reward $J_{\bar{\mathbf{G}}_{\text{best}}}^*(M, L)$.

Theorem 4 *For every partition \bar{V} of G*

$$J_{\bar{\mathbf{G}}_{\text{worst}}}^*(M, L) \leq J_{\mathbf{G}}^*(M, L) \leq J_{\bar{\mathbf{G}}_{\text{best}}}^*(M, L) \quad (4.31)$$

The proof of the lower bound contains a procedure for generating an approximately optimal team search path on \mathbf{G} whose total reward lies between $J_{\bar{\mathbf{G}}_{\text{worst}}}^*(M, L)$ and $J_{\mathbf{G}}^*(M, L)$.

Proof: To verify the lower bound, we use the optimal worst-case team path $\bar{\mathbf{p}}^* = (\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \dots, \bar{\mathbf{v}}_\ell)$ on $\bar{\mathbf{G}}_{\text{worst}}$ to construct a feasible team path $\hat{\mathbf{p}}$ on \mathbf{G} such that $\mathbf{C}(\hat{\mathbf{p}}) \leq L$. First, let us expand the team path to show the paths of each

agent,

$$\bar{\mathbf{p}}^* = \begin{pmatrix} \bar{\mathbf{p}}_1^* \\ \bar{\mathbf{p}}_2^* \\ \vdots \\ \bar{\mathbf{p}}_s^* \end{pmatrix} = \begin{pmatrix} (\bar{\mathbf{v}}_1(1), \bar{\mathbf{v}}_2(1), \dots, \bar{\mathbf{v}}_\ell(1)) \\ (\bar{\mathbf{v}}_1(2), \bar{\mathbf{v}}_2(2), \dots, \bar{\mathbf{v}}_\ell(2)) \\ \vdots \\ (\bar{\mathbf{v}}_1(M), \bar{\mathbf{v}}_2(M), \dots, \bar{\mathbf{v}}_\ell(M)) \end{pmatrix}$$

Now, we begin constructing the lower-level paths $\hat{\mathbf{p}}(a)$ by setting $\hat{\mathbf{p}} = \bar{\mathbf{p}}^*$ and then replacing the meta-vertices $\bar{\mathbf{v}}_i(a)$ with the paths computed by solving the worst-case cooperative search problem on the corresponding subgraphs $G|\bar{\mathbf{v}}_i(a)$. This involves grouping any agents occupying the same meta-vertex and assigning their paths based on the optimal team-path on that meta-vertex with the appropriate occupancy \bar{o} . For each agent a ,

$$\begin{aligned} \hat{\mathbf{p}}_a = & (\mathbf{p}_a^*(\bar{\mathbf{v}}_1(a), \bar{o}), \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_2(a), \bar{o}), \dots, \\ & \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_{\ell-1}(a), \bar{o}), \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_\ell(a), \bar{o})). \end{aligned}$$

Because of (4.26), the sum of the costs of these disconnected sub-paths in $\hat{\mathbf{p}}$ is equal to the sum of the vertex costs in $\bar{\mathbf{p}}^*$. Also, the total reward collected on these sub-paths is equal to the total reward collected on $\bar{\mathbf{p}}^*$ due to (4.25), so we know that $J_{G_{\text{worst}}}^*(M, L) = \mathbf{R}(\bar{\mathbf{p}}^*) \leq \mathbf{R}(\hat{\mathbf{p}})$. We now fill in the gaps between consecutive sub-paths $\mathbf{p}_a^*(\bar{v}, \bar{o})$ by connecting the last vertex in each previous sub-path to the first vertex in the next sub-path with the shortest path in G between them. We know from (4.27) that the cost of these connections is less than or equal to the edge costs in $\bar{\mathbf{p}}^*$. Hence, $C(\hat{\mathbf{p}}) \leq C(\bar{\mathbf{p}}^*) \leq L$ and $\hat{\mathbf{p}}$ is a feasible path

in \mathbf{G} . Since \mathbf{p}^* is optimal on \mathbf{G} , $\mathbf{R}(\hat{\mathbf{p}}) \leq J_G^*(M, L)$, and the lower bound in 4.31 holds.

To verify the upper bound, we construct a feasible path $\tilde{\mathbf{p}}$ in $\bar{\mathbf{G}}_{\text{best}}$ out of the optimal search path $\mathbf{p}^* = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_\ell)$ in \mathbf{G} that generates reward $J_G^*(s, L)$. We begin by setting $\tilde{\mathbf{p}}$ equal to \mathbf{p}^* and then replacing each $\mathbf{v}_i(a)$ with the $\bar{\mathbf{v}}_i(a)$ that contains it. Now, we remove any consecutive repetitions from each $\tilde{\mathbf{p}}_a$, and then pad the end of agent's paths with repeated meta-vertices where necessary to make the paths of all agents the same length. This allows us to form the team-vertices that make up $\tilde{\mathbf{p}}$, making a feasible path in $\bar{\mathbf{G}}_{\text{best}}$ without adding to the path-cost. We infer from (4.28) and (4.30) that $C(\tilde{\mathbf{p}}) \leq C(\mathbf{p}^*)$, so $\tilde{\mathbf{p}}$ meets the cost constraint. Finally, due to (4.29), all reward is collected from each meta-vertex visited by $\tilde{\mathbf{p}}$, and the upper bound $J_G^*(M, L) \leq J_{\bar{\mathbf{G}}_{\text{best}}}^*(M, L)$ holds.

□

4.6.2 Tightness of Approximation for Lattice Graphs

Here we investigate the accuracy of the fractal decomposition algorithm for the search problem on lattice graphs. Suppose a single agent is searching a square lattice graph $G(n) := L^{n \times n}$, with cost bound γn . The vertices all have zero cost and a reward of one and all the edge costs are one. The following lemma shows that in this case, the fractal decomposition algorithm results in an approximation that is asymptotically tight as the number of vertices increases to infinity.

Lemma 6 *For 2-dimensional square lattice graphs of size $n = i^2 \times i^2$, $\forall i \in$*

$\mathbb{N} \setminus \{1\}$,

$$\lim_{n \rightarrow \infty} \frac{J_{G_{\text{worst}}(n)}^*(1, \gamma n)}{J_{G(n)}^*(1, \gamma n)} = 1. \quad (4.32)$$

Proof: On this lattice graph, the optimal reward is $\gamma n + 1$, which the searcher can collect in many ways, for example, by moving along the length of the first row and then proceeding row-by-row until the cost bound has been reached. We now apply the fractal decomposition algorithm by partitioning the graph evenly into \sqrt{n} square subgraphs each containing \sqrt{n} vertices and choosing a meta-vertex cost bound of \sqrt{n} . The lower bound on the optimal search reward, determined by the reward collected on the worst-case meta-graph, is

$$\sqrt{n} \left\lfloor \frac{\gamma n}{\sqrt{n} + 3n^{\frac{1}{4}} - 3} \right\rfloor,$$

where the term \sqrt{n} on the left indicates the reward collected in each meta-vertex, and the term on the right is the conservative worst-case estimate of the number of meta-vertices the searcher can visit. The resulting ratio of worst-case reward to optimal reward is

$$\frac{\sqrt{n} \left\lfloor \frac{\gamma n}{\sqrt{n} + 3n^{\frac{1}{4}} - 3} \right\rfloor}{\gamma n - 1}.$$

Taking the limit as $n \rightarrow \infty$, this ratio approaches one. \square

This extends recursively for multiple decomposition levels. Intuitively, we have this result because the cost incurred in searching a meta-vertex grows as n^2 while the cost incurred in transit between meta-vertices only grows as fast as n .

We conclude that for large graphs, where the transit cost between meta-vertices is small compared to the cost of searching a meta-vertex, the lower-bound on the search reward may be very close to optimal.

4.6.3 Cooperative Search Test Results

We now apply the methods described above to a test case, simulating the search for an object in a large building with many rooms. Figure 4.4 shows a model of the third floor of Harold Frank Hall at UCSB, where a known initial probability distribution for the object is indicated by the shaded regions (dark represents high probability). The floor has been divided into 646 cells, each about 4 square meters in size. There is a graph vertex on each cell and pairs of vertices lying on adjacent cells are connected by an edge in the graph. We assign each edge a cost of 1, modeling a one second transit time between cells, and each vertex a cost of 2, supposing that it takes 2 seconds to search a cell.

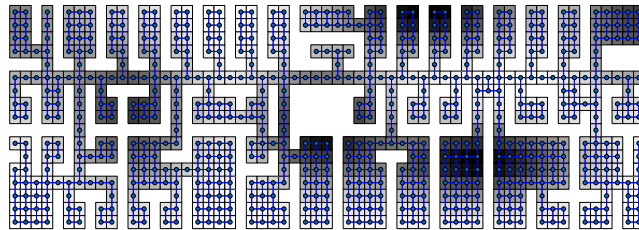


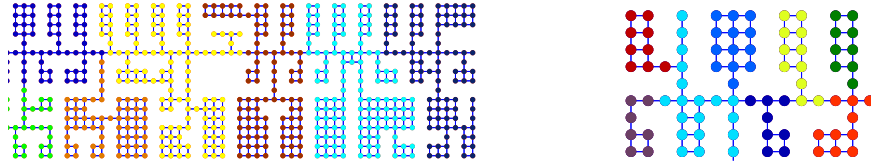
Figure 4.4. Model of third floor of UCSB’s Harold Frank Hall divided into 646 cells and overlaid with a graph. Dark cells indicate high target probability.

In this test case, we have 4 searchers and only two minutes to find the object. The goal is to find the path for each agent that approximately maximizes the probability of finding the object in 120 seconds. To estimate the magnitude of

computation posed by this problem, consider a solution by total enumeration of feasible paths. The average degree of a vertex in this graph is about 3, meaning that the average degree of a vertex in the team graph is $3^4 = 81$, that is, there are about 81 possible moves for the team at each vertex. A cost bound of 120 allows the searchers to visit up to 40 vertices along their paths. This translates to roughly $81^{40} \approx 10^{76}$ paths that must be evaluated to find the optimal search path by total enumeration.

We now apply the fractal decomposition method to this problem. Using two levels of decomposition, we partition the top level into 7 groups and each of the lower-levels into 8, because there are roughly 56 rooms on the floor. We use the automated graph partitioning algorithm in [14], which tries to minimize the total cost of cut edges by clustering the eigenvectors of a (doubly stochastic) modification of the edge-cost matrix around the k most linearly independent eigenvectors. For our purposes, we define the cost of cutting an edge between adjacent vertices with rewards r_1 and r_2 to be $e^{-|r_1-r_2|}$. This causes the algorithm to favor cutting edges with very different rewards, and thus grouping vertices with similar rewards. Figure 4.5(a) shows the top-level partition on our test graph, with vertices of the same color belonging to the same partitioned subgraph. Figure 4.5(b) shows the second-level partition applied to the subgraph in the upper left corner of Figure 4.5(a). The remaining subgraphs are similarly partitioned.

We choose a cost bound allocation of 25 seconds on each of the 56 lower-level subgraphs and 120 seconds on the 7 top-level subgraphs. The maximum vertex occupancy on all levels is set to 1. Now we are ready to run the algorithm. Figure 4.6 shows the approximately optimal paths computed for four searchers. The cost of these paths is 110 seconds, and the searchers collect a reward of 0.29,



(a) Top-level partition on the graph into 7 subgraphs. (b) The subgraph in the upper-left corner of the graph to the left, partitioned on a second level into 8 sub-subgraphs.

Figure 4.5. Two levels of partitioning on the search graph.

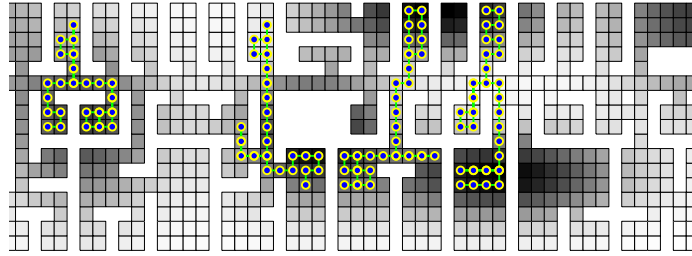


Figure 4.6. Results of 4-agent cooperative search simulation.

which lies between the worst-case lower bound of 0.26 and best-case upper bound of 1.0. Table 5.2 shows the results of the algorithm for one to four searchers. The

Table 4.5. Results of Cooperative Search for 1-4 Agents

Searchers (M)	Cost	Reward	$J_{G_{\text{worst}}}^*(M, 120)$	$J_{G_{\text{best}}}^*(M, 120)$
1	107	0.081	0.072	1.0
2	107	0.15	0.14	1.0
3	110	0.22	0.20	1.0
4	110	0.29	0.26	1.0

fact that M searchers are able to collect almost M times the reward of 1 searcher shows that this algorithm achieves good cooperation between agents. In all four tests, the best-case upper bounds are equal to the total reward contained in the graph. This is not ideal, but when using multiple decomposition levels, it is

difficult to avoid a very optimistic upper bound unless the meta-edge costs are significantly larger than the meta-vertex costs. This is one potential direction for future research.

There is also some backtracking along the paths in Figure 4.6, some of which could be eliminated with a simple algorithm implemented in post-processing. Once this is done, there will be some unused cost available, and the path could be further improved with a greedy algorithm, for example.

Although the initial searcher positions were not fixed in this example, it is straight-forward to apply this algorithm to a problem where they are fixed, by preselecting the initial (meta-)vertices in the top-level search paths as well as those for paths on any subgraphs where the searchers are initially located.

Chapter 5

Graph-Based Receding Horizon Optimization

In the previous chapter, we showed that the fractal decomposition method is useful for approximating a broad class of graph optimization problems in which the data for the problem is static. Now we consider an approach to deal with dynamic problems in which the data changes over time. Specifically, we focus our attention on the computationally challenging problem of cooperative search for mobile targets.

In, receding horizon optimization, one tries to find close-to-optimal solutions to a possibly infinite horizon dynamic problem by solving a shorter, finite horizon problem at each time step. The graph-based receding horizon optimization method presented here differs from conventional receding horizon methods in that the prediction horizon length is defined as a number of graph nodes rather than a length of time. With an appropriate metric for evaluating paths over this hori-

zon, this allows comparison between paths of varying duration. One advantage gained by doing this is that we may optimize on graphs with non-uniform edge lengths, for example, those generated by the sampling algorithms of Chapter 3. This method is designed to significantly reduce computation with relatively low loss of optimality.

5.1 Method

A natural way to reduce the computation involved in dynamic optimization problems is to restrict the optimization to a finite, receding horizon. As mentioned in Chapter 1, there is some literature on receding horizon search algorithms, but only with a fixed time horizon. Since these algorithms require the solution of an NP-hard search problem at each time step, the prediction horizon must be kept short to ensure computational feasibility. This means that if there are regions of high target probability separated by a distance that is much greater than an agent can cover on this horizon, the algorithm’s performance will suffer.

We address this issue by performing the receding-horizon optimization on a dynamically changing graph whose nodes are carefully placed at locations in the search region having the highest target probability. The prediction horizon is defined as a fixed number of waypoints, and since the edges of the graph are not uniform, the algorithm chooses paths that maximize the probability of finding the target per unit cost. This dynamic graph structure facilitates two key strengths of this algorithm: (i) search agents only perform detailed searches in regions with high target probability, and (ii) long paths can be efficiently evaluated and fairly compared to short paths. We call this algorithm the Cooperative Graph-Based

Model Predictive Search Algorithm (CGBMPS).

5.2 Search Problem Formulation

Suppose a team of M agents is searching for a mobile target in a bounded planar region $\mathcal{R} \subseteq \mathbb{R}^2$. Each agent is equipped with a gimballed sensor that it can aim at a limited area of the search region. The region that a sensor can view at a particular time instant is called the *field of view (FOV)*, and the subset of \mathcal{R} viewable by the sensor as it is swept through its entire range of motion (while the agent on which the sensor lies is stationary) is called the sensor's *field of regard (FOR)* (See Figure 5.1).

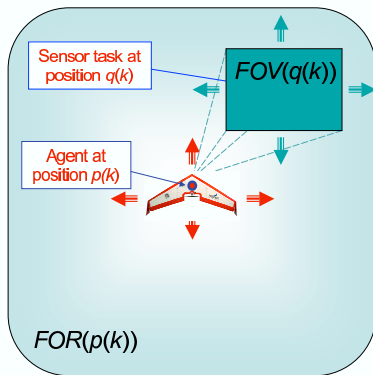


Figure 5.1. Diagram of a field of regard (FOR) and a field of view (FOV).

Let $\mathcal{A} \subseteq \mathbb{R}^3$ be the space in which the agents move. We denote the state of the search team by $\mathbf{p}(k) := [p_1(k), p_2(k), \dots, p_M(k)]$, where $p_a(k) \in \mathcal{A}$ is agent

a 's position at time k , and $k \in \mathbb{Z}_{\geq 0}$ is a discrete time variable belonging to the nonnegative integers. We assume the agents can move in any direction with unit velocity. For each agent, we define a *sensor task* $q_a(k) \in \mathcal{R}$ that specifies where agent a will point its sensor at time k , and we denote the vector of sensor tasks for the team by $\mathbf{q}(k) := [q_1(k), q_2(k), \dots, q_M(k)]$. For simplicity of notation, we assume that the point $q(k)$ at which a sensor is aimed uniquely determines the sensor's field of view. Denoting the set of all possible sensor tasks by \mathcal{Q} , and the set of all subsets of \mathcal{R} by $\mathcal{P}(\mathcal{R})$, we define a function $FOV : \mathcal{Q} \rightarrow \mathcal{P}(\mathcal{R})$ that maps a point at which a sensor is aimed to the subset of \mathcal{R} in view of that sensor. Similarly, we will use the function $FOR : \mathcal{A} \rightarrow \mathcal{P}(\mathcal{R})$ to denote the subset of the search region lying in the field of regard of an agent.

In order to optimize how the agents and sensors should move, we need a mechanism to estimate the probability distribution of the target's location in \mathcal{R} . Specific estimation methods are treated in references [1] and [16]. However, to provide a more general approach, we use the concept of a *target state estimate*, denoted by $\mathbf{x}(k)$, which consists of position and weight components $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$. Each of these two components is an n -vector. The target state estimate may take the form of a particle filter, grid-based probabilistic map, or some other type of estimator. The key requirements we enforce on \mathbf{x} are

$$x_i^p(k) \in \mathcal{R} \quad \forall i \in \{1, \dots, n\}, \forall k, \quad (5.1)$$

$$x_i^w(k) \geq 0 \quad \forall i \in \{1, \dots, n\}, \forall k, \quad (5.2)$$

$$\sum_{i=1}^n x_i^w(k) = 1 \quad \forall k. \quad (5.3)$$

Condition (5.1) ensures that the target state estimate stays inside the search

region, condition (5.2) requires the weights to be nonnegative, and condition (5.3) requires the weights to be normalized at each time step. The initial target state estimate $\mathbf{x}(0)$ is based on any prior knowledge of target location, *e.g.* an initial probability distribution. Future target state estimates depend on sensor tasks $\mathbf{q}(k)$ and previous target state estimates, and can be expressed by a dynamic model of the following form:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{q}(k)), \quad (5.4)$$

where the function f should preserve conditions (5.1)-(5.3). In Sections 5.2.1 and 5.2.2, we show what the dynamics of (5.4) are for a particle filter and a probabilistic map.

We denote by $r(k)$ an instantaneous *team search reward* that effectively measures the probability that the team will discover the target at time k . We say that the target has been *discovered* if it lies within the field of view of at least one agent and one of those agents detects its presence. The initial value for the team search reward is $r(0) = 0$, and future values depend on the target state estimate $\mathbf{x}(k)$, the sensor tasks $\mathbf{q}(k)$, and previous rewards according to an expression of the following form:

$$r(k+1) = g(\mathbf{x}(k), \mathbf{q}(k)) \left(1 - \sum_{\kappa=0}^k r(\kappa)\right), \quad (5.5)$$

where the value of the function g corresponds to the conditional probability that the target will be found at time $k+1$, given that it was not previously found. The term multiplying $g(\cdot)$ corresponds to the probability that the target was not

found at or before time k . Therefore, $r(k+1)$ indeed measures the probability that the target will be found at time $k+1$.

It will be convenient to have an expression for the total reward collected by the team as a summation of the rewards collected by each individual agent. With this in mind, we rewrite r as

$$r(k+1) = \sum_{a=1}^M r_a(k+1),$$

where r_a is an instantaneous *agent search reward*, which we can express by

$$r_a(k+1) = g_a(\mathbf{x}(k), \mathbf{q}(k)) \left(1 - \sum_{\kappa=0}^k r(\kappa)\right), \quad (5.6)$$

for each agent $a \in \{1, \dots, M\}$. Here, the function g_a measures the conditional probability that agent a will find the target at time $k+1$ given that it was not previously found. A summation over the g_a 's yields the function g used in (5.5):

$$g(\mathbf{x}, \mathbf{q}) = \sum_{a=1}^M g_a(\mathbf{x}, \mathbf{q}).$$

Using the target weights \mathbf{x}^w , we can explicitly write the function g_a as

$$g_a(\mathbf{x}, \mathbf{q}) := \sum_{i=1}^n x_i^w \rho_a(x_i^p, \mathbf{q}),$$

where $\rho_a(x, \mathbf{q})$ is an *agent reward factor*, which measures the conditional probability that, when the team's sensors are aimed at the set of points \mathbf{q} , agent a will find the target at the point x given that the target is located at x . Assuming that each agent correctly detects a target located inside its sensor's FOV with

probability $P_D \in (0, 1]$, we can express the agent reward factor as

$$\rho_a(x, \mathbf{q}) := \delta_v(x, q_a) P_D \prod_{\alpha=1}^{a-1} (1 - \delta_v(x, q_\alpha) P_D), \quad (5.7)$$

where the function $\delta_v(x, q)$ indicates whether a particular point x is in view of a sensor aimed at the point q , and is given by

$$\delta_v(x, q) := \begin{cases} 1, & x \in FOV(q) \\ 0, & \text{otherwise} \end{cases}.$$

Although we have introduced in (5.7) a specific ordering to the agents for the purposes of computation, one can show that changing this order does not affect the value of g .

5.2.1 Example 1: Particle Filter

Particle filtering is a Monte Carlo-based method for state estimation that is especially well-suited to systems with non-linear dynamics and non-Gaussian distributions. It involves modeling a system with a large number of dynamic “particles” whose states evolve according to some stochastic model. Each particle is assigned a weight representing the likelihood that the actual state is near that particle. Weights are typically updated upon the arrival of new measurements and are then generally normalized so that the total weight of the particles is equal to one. See [1] for a more detailed treatment of particle filtering. We now give an example of how to implement (5.4) using particle filter estimation.

Let $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$ represent the states of a simple particle filter, where

each $x_i^p(k)$ is the position of the i^{th} particle and $x_i^w(k)$ its weight. The system dynamics are then given by

$$\begin{aligned}\tilde{x}_i^w(k+1) &= x_i^w(k) \prod_{a=1}^M (1 - \delta_v(x_i^p(k), q_a(k)) P_D) & \forall i, \\ \mathbf{x}^w(k+1) &= \tilde{\mathbf{x}}^w(k+1) \frac{1}{\sum_{i=1}^n \tilde{x}_i^w(k+1)}, \\ x_i^p(k+1) &= f_p(x_i^p(k), w_i(k)),\end{aligned}$$

where $\tilde{\mathbf{x}}^w$ is an intermediate vector of unnormalized target weights, and $\mathbf{w}(k) := [w_1(k), \dots, w_n(k)]$ is a process noise sequence. The function f_p is typically used to propagate the particle positions according to a model of the expected target dynamics, randomized with $\mathbf{w}(k)$, which for our purposes is fixed for the duration of the search and known by all agents.

5.2.2 Example 2: Grid-based Probabilistic Map

A grid-based probabilistic map is another method for estimating the states of an uncertain system. It involves representing a system's state-space by a grid of cells, each of which has a weight corresponding to the probability of the target being located inside that cell. The cell weights are updated based on incoming measurements and predicted future states. See [16] for more on probabilistic maps. We now give an example of how to implement (5.4) using a probabilistic map.

Let $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$ represent the states of the cells in the map. Here, each $x_i^p(k)$ is the static position of the center of cell i and each $x_i^w(k)$ is its dynamic

weight. In this case, the system dynamics are

$$\begin{aligned}\tilde{x}_i^w(k+1) &= x_i^w(k) \prod_{a=1}^M (1 - \delta_v(x_i^p(k), q_a(k)) P_D) & \forall i, \\ \hat{\mathbf{x}}^w(k+1) &= f_w(\tilde{\mathbf{x}}(k+1)), \\ \mathbf{x}^w(k+1) &= \hat{\mathbf{x}}^w(k+1) \frac{1}{\sum_{i=1}^n \hat{x}_i^w(k+1)},\end{aligned}$$

where where $\tilde{\mathbf{x}}^w$ and $\hat{\mathbf{x}}^w$ are intermediate vectors of target weights, and the function f_w can be used to diffuse target weights between adjacent cells according to a transition probability matrix.

5.2.3 Problem Statement

We define a *search policy* to be a function μ that maps the current team state $\mathbf{p}(k)$ and target state estimate $\mathbf{x}(k)$ to the next set of controls to be executed by the team, that is, each agent's next sensor task is given by $q_a(k) = \mu(\mathbf{p}(k), \mathbf{x}(k))$. The goal of the search problem is to find the search policy that results in the fastest possible discovery of the target by the team.

Let T^* denote the time at which the target is discovered by any agent. This time is unknown at the beginning of the search, but we can write its expected value as

$$E_\mu[T^*] = \sum_{k=0}^{\infty} kr(k). \quad (5.8)$$

We can now express the objective of the search problem as

$$\min_{\mu} E_{\mu}[T^*] \tag{5.9}$$

subject to

$$FOV(q_a(k)) \subseteq FOR(p_a(k)) \quad \forall a, k. \tag{5.10}$$

The constraint (5.10) requires each agent’s sensor task to be feasible with respect to its current field of regard. This constraint is needed because of the physical constraints of the gimballed sensors.

For agents with such sensor and motion constraints, the problem posed in (5.9) is a difficult combinatorial optimization problem. Even for the case of a single agent, stationary target, and fixed sensor, this search problem is known to be NP-Hard [30]. Our approach consists of approximating the solution by choosing paths on a graph that maximize the reward collected over a finite, receding horizon. This method is designed to reduce computation by optimizing over a shorter time horizon on a much smaller state space, while also using careful vertex placement to maintain route flexibility in regions of high reward. Although our algorithm does not necessarily result in a policy that minimizes $E[T^*]$, the resulting policy does lead to a finite value for this expected time and we provide an upper bound for this quantity in Section 5.4.

5.3 Cooperative Graph-based Model Predictive Search Algorithm

In this section, we present a receding horizon search algorithm that plans paths and sensor tasks for a team of agents based on predicted future target states. We use a prediction horizon based on waypoints rather than time because we would like to consider paths of varying duration. This allows agents to travel between high-reward regions separated by large areas with low reward.

5.3.1 Graph Search Formulation

Let $G := (V, E)$ be a graph for path-planning with vertex set V and edge set E . The vertices of the graph will serve as waypoints for the agents, which are connected by graph edges to form paths. When there is no ambiguity, we will use the notation v to denote both the vertex in the set V and its spatial location in \mathcal{A} . Similarly, $e = (v, v')$ will denote both the edge in the set E and the set of points on the line segment connecting its endpoint vertices v and v' . Lastly, we define an edge-cost function $c : E \rightarrow [0, \infty)$, representing the transit time between vertices, assuming the agents move with unit velocity.

In the CGBMPS algorithm, whenever an agent reaches a waypoint, it will compute a new ℓ -step path on the graph, where ℓ is the length of the prediction horizon. A *path* in G is a sequence of vertices $P := (v_1, v_2, \dots, v_{\ell-1}, v_\ell)$ such that

$(v_i, v_{i+1}) \in E$. The *path cost* is the total duration of the path and is given by

$$C(P) := \sum_{i=1}^{\ell-1} c(v_i, v_{i+1}).$$

The reward for a path P will depend on the expected sensor coverage along that path.

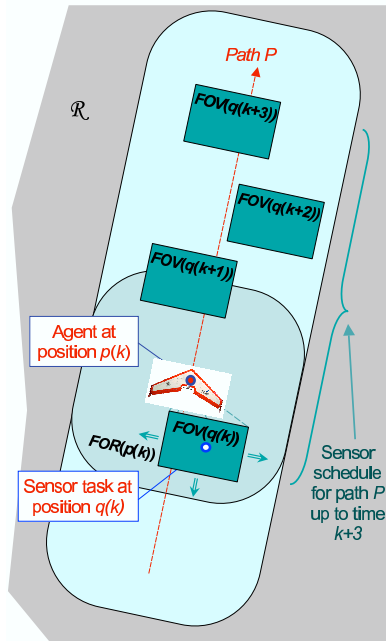


Figure 5.2. Example sensor schedule for an agent.

In an attempt to minimize the expected time $E[T^*]$ to find the target, each agent will select the path from the set of all possible ℓ -step paths, that maximizes the team's probability of finding the target. We will shortly define a *path reward* corresponding to this probability, and a *path cost* corresponding to the duration of a path. In order to compare paths of varying length, the algorithm uses an optimization criterion based on path reward per unit cost. As each agent travels

along its path P , it executes a sequence of sensor tasks, which we call a *sensor schedule* and denote by $S(P, k) := (q(k), q(k+1), \dots, q(k+T_P))$, where each $q(\cdot)$ is a scheduled sensor task and $T_P := \lfloor C(P) \rfloor$ is the total number of sensor tasks that may be executed along the path P . In this discrete time setting, we assume that the duration of each sensor task is one unit of time. Figure 5.2 shows a diagram of an agent executing a sensor schedule along its path P . The sensor tasks $q(k)$ may be chosen arbitrarily or computed by another optimization. This allows for several possible methods of algorithm implementation, including

1. **Fixed Sensor Tasking:** Construct $S(P, k)$ assuming a fixed pattern of sensor motion, such as slewing side to side within the field of regard.
2. **Joint Routing and Sensor Optimization:** Construct $S(P, k)$ by optimizing over all possible sensor schedules along the path P .

Fixed sensor tasking requires much less computation, but joint routing and sensor optimization will generally yield better results. For situations in which the relative speed of the agents is too fast for the sensors to view everything inside the agents' FORs, method 2 may yield significant benefit over method 1. We will revisit these methods in Section 5.5, but for the remainder of this section, we will assume that we are given an algorithm to compute the sensor schedule $S(P, k)$ for a given path P .

For a path P_a , we define the *path reward* for agent a as

$$R_a(P_1, \dots, P_a) := \sum_{\kappa=k}^{k+T_{P_a}} r_a(\kappa), \quad (5.11)$$

where $r_a(\kappa)$ is computed using (5.6) with the sensor tasks $q_i(\kappa)$ from the sensor

schedules $S(P_i, k)$ obtained from the paths P_i of agents $i \in \{1, \dots, a\}$. Note that the paths P_i of agents $i \in \{a + 1, \dots, M\}$ do not affect (5.11). To provide a fair measure of reward over paths of different lengths, we define the *normalized path reward* for the path P_a as

$$\bar{R}_a(P_1, \dots, P_a) := \frac{R_a(P_1, \dots, P_a)}{C(P_a)}. \quad (5.12)$$

As mentioned previously, each agent computes a new path whenever it reaches a waypoint. Since the graph edges have nonuniform length, the agents will generally arrive at waypoints, and hence compute paths, at different times. At each time k , we therefore define two groups of agents: the set $A_{\text{plan}}(k)$ of agents that have arrived at waypoints and need to plan new paths, and the set $A_{\text{en}}(k)$ of the remaining agents that are en route to waypoints. For simplicity of notation, we assume that the indexing of agents $\{1, \dots, M\}$ is given by the sequence $\{A_{\text{en}}(k), A_{\text{plan}}(k)\}$ at each time k . In the CGBMPS algorithm, whenever the set $A_{\text{plan}}(k)$ is nonempty, the agents in that set compute new paths assuming that the agents in the set $A_{\text{en}}(k)$ will continue on their current paths. With this formulation, we can express each step in the receding horizon optimization as follows:

$$\{P_a^* : a \in A_{\text{plan}}(k)\} := \arg \max_{\{P_a : a \in A_{\text{plan}}(k)\}} \sum_{i=1}^M \bar{R}_i(P_1, \dots, P_i). \quad (5.13)$$

Often, the set $A_{\text{plan}}(k)$ consists of a single agent, reducing (5.13) to an optimization for the path of that agent. If this is not the case, a computationally practical approximation to (5.13) consists of a sequential optimization by the

planning agents, which we can express as

$$P_a^* := \arg \max_{P_a} \bar{R}_a(P_1^*, \dots, P_{a-1}^*, P_a), \quad \forall a \in A_{\text{plan}}(k), \quad (5.14)$$

where each P_i^* is either the current path of an en route agent $i \in A_{\text{en}}(k)$ that was computed at a previous time or a path computed using (5.14) by an agent earlier in the sequence $A_{\text{plan}}(k)$.

5.3.2 Dynamic Graph Generation

Although the CGBMPS algorithm will work with any graph, the structure of the graph will have a major effect on computation, and the vertex and edge placements will affect the algorithm's performance. Also, since the target state estimate is constantly changing due to sensor measurements and predicted target motion, the graph should be periodically updated to allow the agents to search new high-reward areas that were low-reward areas in previous time steps. This graph update should occur at each time k at which the set of agents $A_{\text{plan}}(k)$ is nonempty. We now present a dynamic graph construction process that will guarantee some performance properties of the CGBMPS algorithm. Figure 5.3 shows an example of this process.

In the following procedure, we will use the function $W(\mathbf{x}, e)$ to denote the sum of the weights of all components of the target state estimate lying inside the FOR of some point on the edge e . We define this as

$$W(\mathbf{x}, e) := \sum_{i=1}^n x_i^w \delta_r(x_i^p, e),$$

where $\delta_r(x, e) \in \{0, 1\}$ indicates whether or not the point x_i^p is inside the FOR of at least one point on the edge e , and is expressed by

$$\delta_r(x, e) := \begin{cases} 1, & x \in \bigcup_{p \in e} FOR(p) \\ 0, & \text{otherwise} \end{cases}.$$

Graph Construction Process

1. Construct a uniform lattice graph $G := (V, E)$ over the search region \mathcal{R} having the property that for every point r in \mathcal{R} , there exists an edge $e \in E$ such that the point r falls within the FOR of some point along the edge e . This can always be achieved by choosing a sufficiently small vertex spacing in the lattice [cf. Figure 5.3(a)].
2. Choose a reward threshold $\tau > 0$ and let $G_k^{\text{th}} := (V_k^{\text{th}}, E_k^{\text{th}})$ be the subgraph of G induced by the edge set $E_k^{\text{th}} := \{e \in E : W(\mathbf{x}(k), e) \geq \tau\}$. The vertex set V_k^{th} consists of the union of the set of vertices connected by edges E_k^{th} with the set of vertices that serve as initial waypoints v_1^a for the agents [cf. Figure 5.3(b)].
3. Let $G_k^{\text{Del}} := (V_k^{\text{th}}, E_k^{\text{Del}})$ be the graph generated by a Delaunay triangulation of V_k^{th} [23]. Next, let $G_k^{\text{Del} < d} := (V_k^{\text{th}}, E_k^{\text{Del} < d})$ be the subgraph of G_k^{Del} obtained by keeping only the edges connecting vertices of degree less than d , where d is the degree of the lattice graph G . That is $E_k^{\text{Del} < d} := \{(v, v') \in E_k^{\text{Del}} : \deg(v) < d \text{ and } \deg(v') < d\}$ [cf. Figure 5.3(c)].
4. The final graph $G_k := (V_k, E_k)$ is the combination of graphs G_k^{th} and $G_k^{\text{Del} < d}$, with vertex set $V_k := V_k^{\text{th}}$ and edge set $E_k := E_k^{\text{th}} \cup E_k^{\text{Del} < d}$ [cf. Figure 5.3(d)].

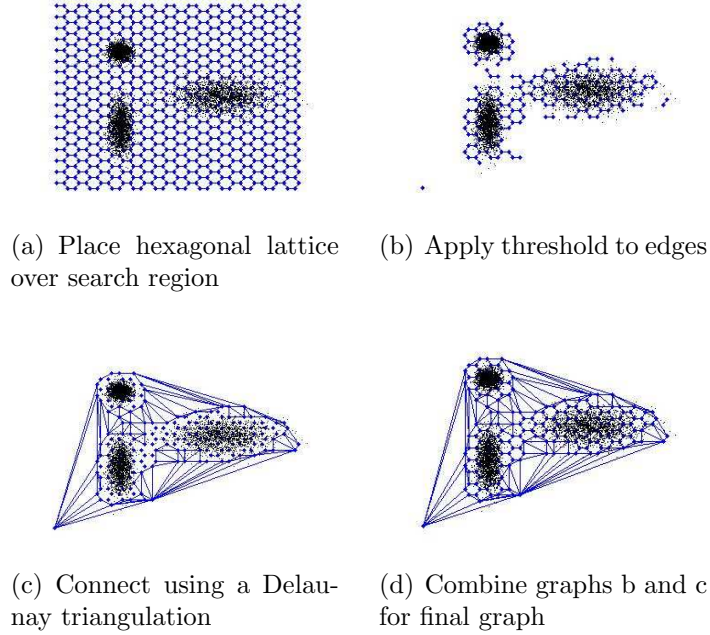


Figure 5.3. Example of graph construction process.

In step 1, any type of lattice graph will suffice, but we have found that a degree-3 hexagonal lattice is a particularly good choice. The threshold τ in step 2 should be carefully chosen so that it includes only high-reward edges but does not exclude so many edges that the number of feasible paths is severely restricted. The purpose of step 3 is to ensure that the graph G_k is connected and provides paths between separated areas of high reward. We include only the Delaunay edges connecting vertices of degree less than 3 because these are the edges between boundary vertices of the disconnected components in the graph. The remaining Delaunay edges are unnecessary. Step 4 combines the graphs of steps 2 and 3 to produce the final graph.

We now state an assumption on the dynamics of the target state estimate that is needed in the proof of Lemma 7. This is effectively a requirement that the total target weight within the FOR of an edge cannot drop below some small

value that is a function of time.

Assumption 1 For every edge $e \in E_k$, there exists a constant $\gamma > 0$ such that for every $m \in \mathbb{N}$, the evolution of the target state estimate $\mathbf{x}(k)$ governed by (5.4) satisfies the property

$$W(\mathbf{x}(k+m), e) \geq \gamma^m W(\mathbf{x}(k), e).$$

For a grid-based probabilistic map, this assumption is satisfied provided that there is at least one nonzero entry in the columns of the transition probability matrix for the cells inside the FOR of points in e . For a particle filter, this assumption is satisfied for most motion models with a sufficiently large number of particles.

Under this relatively mild assumption on the target state estimate, Lemma 7 provides a lower bound on the conditional probability that the target will be found at some time in the interval $[k + 1, k + T]$ given that it was not found at or before time k .

Lemma 7 There exists a constant $\epsilon > 0$ and a time $T > 0$ such that for every graph G_k generated by the Graph Construction Process and every set of ℓ -length paths P_1, \dots, P_M in G_k , there exists a sequence of sensor tasks $(\mathbf{q}(k), \dots, \mathbf{q}(k + T - 1))$ for which $\sum_{\kappa=k}^{k+T-1} g(\mathbf{x}(\kappa), \mathbf{q}(\kappa)) \geq \epsilon$.

Proof: Choose vertex v_2 for which there exists an edge $e \in E_k$ connecting vertices v_1 and v_2 . This is always possible because the Delaunay triangulation guarantees

that G_k is connected. The graph construction process ensures that $W(\mathbf{x}(k), e) \geq \tau$. Let D_{\max} be the maximum Euclidean distance between any two vertices in G_k . Using the fact that agents travel with unit velocity, denote the maximum number of time steps between two vertices by $T := \lfloor D_{\max} \rfloor$. By Assumption 1, we have that $W(\mathbf{x}(k+T), e) \geq \gamma^T W(\mathbf{x}(k), e)$, and thus $W(\mathbf{x}(k), e) \geq \gamma^T \tau$. We now choose a sensor schedule $S(P_a, k)$ that includes the sensor task

$$q(k+m) := \arg \max_{q \in \mathcal{Q}_e} \sum_{i=1}^n x_i^w \delta_v(x_i, q)$$

for some $m \leq T$, where \mathcal{Q}_e is the set of all feasible sensor tasks for an agent along e . The minimum amount of target weight contained within this sensor task is achieved for a uniform target weight distribution, and is given by $\gamma^T \tau P_D(A_{\text{FOV}}/A_{\text{FOR}})$, where A_{FOV} and A_{FOR} are the areas of the FOV and the FOR, respectively. We conclude from (5.6) and (5.11) that $\sum_{a=1}^M R_a(P_1, \dots, P_a) \geq \gamma^T \tau P_D(A_{\text{FOV}}/A_{\text{FOR}})(1 - \sum_{\kappa=0}^k r(\kappa))$. Therefore, Lemma 7 holds with

$$\epsilon = \gamma^T \tau P_D(A_{\text{FOV}}/A_{\text{FOR}})(1 - \sum_{\kappa=0}^k r(\kappa))$$

□

5.3.3 CGBMPS Algorithm

The Cooperative Graph-Based Model Predictive Search algorithm is described in Table 5.1. It begins with initialization of the graph and each agent's path and sensor schedule in steps 2 and 3. At each subsequent time step k , whenever the set of agents $A_{\text{plan}}(k)$ is nonempty, the graph is updated, and the agents in

$A_{\text{plan}}(k)$ select new paths and sensor schedules according to (5.13). This process repeats until the target is found at time T^* , which is proven to be finite with probability one in Theorem 5.

Table 5.1. Cooperative Graph-Base Model Predictive Search Algorithm

Algorithm 1 CGBMPS

```

1:  $k := 0$ 
2: Construct  $G_0 = (V_0, E_0)$  as in Section 5.3.2
3: For each agent  $a \in \{1, \dots, M\}$ , assign an arbitrary initial path  $P_a = (v_1^a, v_2^a)$ 
   on  $G_0$  and a corresponding sensor schedule  $S(P_a, 0)$ 
4: while target has not been discovered do
5:   Each agent  $a$  points its sensor at  $q_a(k)$  and takes measurements
6:   Compute the set of agents arriving at waypoints  $A_{\text{plan}}(k)$ 
7:   if the set  $A_{\text{plan}}(k)$  is nonempty then
8:     Update  $G_k$  as in Section 5.3.2
9:     for each agent  $a$  in the set  $A_{\text{plan}}(k)$  do
10:      Compute the path  $P_a^*$  starting from  $v_2^a$  using (5.13)
11:      Assign a sensor schedule  $S(P_a^*, k)$ 
12:     end for
13:   end if
14:   Agents move toward next waypoint in path
15:   Evaluate  $\mathbf{x}(k+1)$  and  $r(k+1)$  using (5.4) and (5.5)
16:    $k := k+1$ 
17: end while
18:  $T^* := k-1$ 

```

Whenever a path is computed, it starts from the waypoint after the one that was reached. There are two key reasons for implementing the algorithm in this way. First, this allows time for computation of the next path, avoiding situations where the agent has reached a waypoint and does not have any destination until it completes a computation. Second, it is practical to have one waypoint planned in advance so that sharp turns in the upcoming path may be smoothed.

5.3.4 Computational Complexity

The CGBMPS algorithm provides a computationally efficient method for approximating the original search problem posed in (5.9) by using a graph that is designed to allow agents to optimize over a set containing only the most rewarding paths. Let K_e denote the maximum time required to compute the reward collected along an edge e in the graph. For example, K_e may represent an upper bound on the computation time required by *fixed sensor tasking* or *joint routing and sensor optimization*, as described in Section 5.3.1. Performing an exhaustive search over this set of paths results in a bound on the computation time that depends on the maximum degree of the graph d , the length of the prediction horizon ℓ , and the number of agents M , by the expression $M!K_e\ell d^\ell$. Note that this is a worst-case computation bound because in a non-uniform graph, it is quite rare that all agents will arrive at waypoints simultaneously and hence compute new paths all at the same time. The factor $M!$ corresponds to the optimization given in (5.13), in which an exhaustive search is performed over every possible order of agents. If one opts to use the computationally simpler, sequential approximation (5.14) instead, the expression becomes $MK_e\ell d^\ell$. Additionally, in the computation of the optimal path P_a^* , we can take advantage of the property that the reward for the paths (v_1, v_2, v_3) and (v_1, v_2, v_4) is the same between vertices v_1 and v_2 . The fact that one only needs to compute the reward for the segment (v_1, v_2) once, further reduces the total computation required for this algorithm. This results in the slightly improved bound of $MK_e \sum_{i=1}^{\ell} d^i$ to compute paths for the team of agents. It is clearly computationally advantageous to keep the degree d of the graph low and the prediction horizon ℓ short. One can also reduce the total number paths to evaluate by doing some reasonable pruning on the set of

candidate paths \mathcal{P}_a , such as eliminating backtracking and paths with sharp turns that the agents cannot physically follow. Here, we have shown computation results for an exhaustive search over the prediction horizon, but it should be noted that algorithms such as branch and bound [6] may further reduce complexity.

5.4 Persistent Search

In this section we show that the CGBMPS algorithm results in finite-time target discovery with probability one. We use the notion of a *persistent search policy*, which is inspired by the persistent pursuit policies discussed in [15].

A search policy μ as defined in Section 5.2.3 is said to be *persistent* if there exists some $\epsilon > 0$ such that

$$g(\mathbf{x}(k), \mathbf{q}(k)) > \epsilon \quad \forall k \in \mathbb{Z}^{\geq 0}, \quad (5.15)$$

where the sensor tasks $\mathbf{q}(k)$ are generated by the search policy μ . In other words, the probability of locating the target at time k given that it was not found previously must always be greater than ϵ . This is a difficult property for a search policy to satisfy in general, but if we know that the agents will collect positive reward over some finite time horizon, we can guarantee the following slightly weaker property. A search policy is said to be *persistent on the average* if there is some $\epsilon > 0$ and some $T \in \mathbb{N}$ such that

$$\sum_{i=0}^{T-1} g(\mathbf{x}(k+i), \mathbf{q}(k+i)) > \epsilon \quad \forall k \in \mathbb{Z}_{\geq 0}, \quad (5.16)$$

where the sensor tasks $\mathbf{q}(k)$ are generated by the search policy μ . The time T is called the period of persistence. Let $F_\mu(k) := \mathbf{P}(T^* \leq k | \mu)$ denote the distribution function of T^* given the search policy μ . We can alternatively write this as

$$F_\mu(k) = \sum_{\kappa=1}^k r(\kappa) = 1 - \prod_{\kappa=1}^k (1 - r(\kappa)).$$

The following lemma is proved in [15].

Lemma 8 For a persistent on the average search policy μ with period T , $\mathbf{P}(T^* < \infty | \mu) = 1$, $F_\mu(k) \geq 1 - (1 - \epsilon)^{\lfloor \frac{k}{T} \rfloor}$, $\forall k \in \mathbb{Z}_{\geq 0}$, and $E_\mu[T^*] \leq T\epsilon^{-1}$, with ϵ given in (5.16).

We now state the main result on finite-time target discovery, which proves that the CGBMPS algorithm terminates with probability one.

Theorem 5 The CGBMPS algorithm results in target discovery in finite time with probability one and

$$E[T^*] \leq \frac{D_{\max}}{\epsilon},$$

where $\epsilon = \gamma^T \tau P_D(A_{\text{FOV}}/A_{\text{FOR}})$.

From Lemma 7, we conclude that the search policy generated by the CGBMPS algorithm, using optimal sensor schedules, is persistent on the average. Theorem 5 then follows directly from Lemma 8 with ϵ as in Lemma 7.

5.5 Simulations

We now simulate a cooperative search scenario with four agents searching for a target in a 5 km by 5 km square region. The FOR of each agent is a circle 600 m in diameter and its FOV is a circle 150 m in diameter. The agents take measurements every 10 seconds. These parameter values are inspired by a physical system with cameras mounted on UAVs that we have successfully tested. The initial target pdf consists of a weighted sum of four randomly placed Gaussian distributions with random covariances, and the target state estimate is constructed using a particle filter as described in Section 5.2.1. For path planning, we use a degree-3 hexagonal lattice graph, which is dynamically updated using the process described in Section 5.3. The agents use a three-step prediction horizon. Figure 5.4 shows a snapshot of the simulation display.

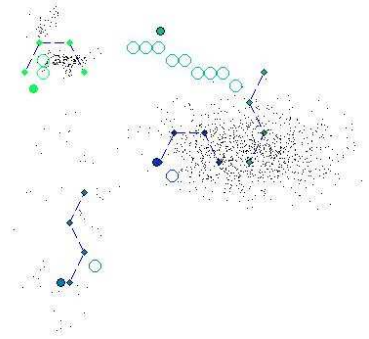


Figure 5.4. Display of Search Simulation.

The agents are represented by the solid circles, and the diamonds are their waypoints. The open circles represent the FOVs of the scheduled sensor tasks on the immediate graph edge. The particles of the particle filter used for target state estimation are represented by the black dots. Only particles with weights

above a certain threshold are displayed.

Table 5.2. Results of Cooperative Search for 1-4 Agents

Number of Agents (M)	Mean Reward (500 Runs)
1	0.22
2	0.39
3	0.53
4	0.64

Table 5.2 shows the results of the CGBMPS algorithm for one to four cooperating agents. The fact that the reward increases significantly with each increase in the number of searchers shows that this algorithm makes effective use of additional agents.

Table 5.3. Algorithm Performance Comparison

Method (4 agents)	Mean Reward (300 Runs)
Greedy search	0.69
3-step fixed RH Search	0.72
CGBMPS (fixed sensor)	0.81
CGBMPS (joint sensor optimization)	0.87

In Table 5.3, we compare the performance of the CGBMPS algorithm with previously proposed search algorithms. The greedy search algorithm in line 1 chooses one-step paths on a static square lattice graph that maximize the probability of finding the target. This is similar to the greedy pursuit policies proposed in [15]. In line 2, we used a three-step receding horizon algorithm on a static square lattice graph, which is similar to the algorithm described in [24]. Lines 3 and 4, when compared to lines 1 and 2, show that the CGBMPS algorithm performs significantly better than the other algorithms that use a fixed time horizon. Additionally, in this scenario, the method of joint routing and sensor tasking performed 7% better than fixed sensor tasking.

The CGBMPS algorithm has also been successfully tested in hardware using a team of two UAVs with cameras to locate a target in a region approximately 2 km by 2 km in size. In each of several tests, the UAVs successfully found a target on the ground within 15 minutes.

Chapter 6

Conclusion

In this work, we have introduced two new approximation algorithms for optimization on large graphs and applied them to cooperative routing problems. When the cooperative routing problem is posed on a continuous domain, we showed how to sample the continuous region to generate graphs for which the optimal value is within ϵ of that of the continuous optimization. The sampling and graph construction processes are key steps in the algorithm and should not be overlooked because the performance of whatever approximation method is used will be limited, or enhanced, by this graph.

The first method, fractal decomposition, is an algorithm that approximates the solutions to large graph optimization problems by partitioning the graph and solving several smaller problems: one for each of the partitioned subgraphs, called meta-vertices, and one for the meta-graph generated by the partition. Each of these problems is in exactly the same form as the original problem, and may therefore be approximated by applying fractal decomposition on a second level.

In this manner, one may recursively decompose the problems on many levels. From the solutions to the meta-problem and each sub-problem, one can construct an approximate solution to the original problem, whose optimal value is guaranteed to be bounded by that of the worst-case meta-problem. Furthermore, we construct a best-case meta-problem, the solution of which provides a measure of how far the approximate solution is from the optimal one. As the number of decomposition levels increases, the computational complexity approaches linearity in the number of vertices at the expense of looser approximation bounds. The worst-case and best-case bounds are problem dependent, but we provide constant factor approximation bounds for the shortest path and maximum flow matrix problems on lattice graphs, which could appear to be poorly suited to hierarchical decomposition due to their uniform structure. For the graph search problem on a lattice, we show that the worst-case bound becomes asymptotically tight as the number of vertices grows very large. For each of the three example problems presented in Chapter 4, we also give numerical simulations to illustrate the accuracy of approximation in practice as well as the computational speed for large problems.

There are several potential directions for future work on the fractal decomposition method. One is to find more applications of this approach. We proposed the fractal decomposition methodology with the intention that it could be applied to a broad class of problems beyond shortest path, maximum flow, and graph search. Problems that may be worth investigating for this approach include pursuit-evasion games, the traveling salesman problem, and various extensions to the cooperative routing problems discussed. For example, in the cooperative search problem, we would like to generalize to the case of a mobile

target, a somewhat more difficult problem because the target probability distribution is constantly changing as time passes and new information is collected. In cooperative routing problems, there is also a potential advantage to be gained by partitioning not only the routing domain, but the search team itself. For instance, a problem involving 40 agents searching a building with 100 rooms might be reduced to solving 4 problems with teams of 10 agents each searching 25 rooms. Then the smaller problems may be further decomposed. This would likely facilitate a larger reduction in computation and, intuitively, would produce reasonably good solutions.

The second method, receding horizon graph optimization, provides an approximation based on a kind of temporal decomposition. In Chapter 5, we proposed a Cooperative Graph-Based Model Predictive Search algorithm that uses a dynamically updated graph and achieves finite-time target discovery with probability one. The key contributions of this algorithm are (1) optimizing on a dynamic graph updated based on changing target probability distribution, (2) using a waypoint-based prediction horizon allowing for comparison between paths of different length, and (3) incorporating the use of gimbaled sensors whose orientations can be jointly optimized with the paths of the agents. Simulations showed that the waypoint-based prediction horizon with a strategic placement of graph vertices causes a significant boost in search algorithm performance over traditional implementations having a fixed time horizon. Using joint routing and sensor optimization provides an additional increase in performance. Furthermore, the CGBMPS algorithm has been successfully tested on a physical system consisting of two UAVs with gimbal-mounted cameras.

An important avenue for future research on receding horizon graph optimiza-

tion is decentralization. One would expect that decentralized policies, in which each agent keeps estimates on the positions and sensor measurements of its teammates, while updating these estimates whenever agents communicate with each other, could produce search paths almost as good as the centralized algorithm but with substantial reduction in communication requirements. Some preliminary analysis of decentralized search policies has shown very promising results. Another aspect of this algorithm that warrants investigation is the method of normalization used when comparing paths of varying cost. The average reward per unit cost is an intuitively appealing solution, but perhaps a weighted sum that favored soon-to-be-collected reward would perform better. More analysis would be required to establish an optimal weighting scheme. Finally, there is the possibility of combining the fractal decomposition and receding horizon approaches for mobile-target search problems. There will be some challenges involved in constructing worst-case and best-case bounds, but such an approach could potentially be very appealing with regard to computation.

Appendix A

Proofs

Lemma 1 *Given any positive constants A , ϵ_x , ϵ_v , and ϵ_δ , there exists a finite convex partition $\{\mathcal{R}_1, \dots, \mathcal{R}_K\}$ of \mathcal{R} and a finite set \mathcal{X} of points in the boundaries of the \mathcal{R}_i such that for every twice continuously differentiable path $\rho : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} with second derivative bounded by A , one can find*

1. *a sequence of times $\{\tau_0, \tau_1, \dots, \tau_N\} \subset [0, T]$, with $\tau_0 := 0 \leq \tau_{k-1} < \tau_k \leq \tau_N := T$, and*
2. *a sequence of points $\{x_0 := \rho(0), x_1, \dots, x_N := \rho(T)\} \in \bar{\mathcal{X}}$, where $\bar{\mathcal{X}} := \mathcal{X} \cup \{\rho(0), \rho(T)\}$,*

such that

$$\|x_k - x_{k-1}\| \leq \epsilon_\delta, \quad \forall k \in \{1, \dots, N\}, \quad (\text{A.1})$$

$$\|\rho(t) - x_{k-1}\| \leq \epsilon_\delta, \quad \forall t \in [\tau_{k-1}, \tau_k], \quad k \in \{1, \dots, N\}, \quad (\text{A.2})$$

$$\|\rho(\tau_k) - x_k\| \leq \epsilon_x, \quad \forall k \in \{0, 1, \dots, N\}, \quad (\text{A.3})$$

$$\left\| \dot{\rho}(\tau) - \frac{x_k - x_{k-1}}{\|\tau - \tau_{k-1}\|} \right\| \leq \epsilon_v, \quad \forall \tau \in (\tau_{k-1}, \tau_k), \quad k \in \{1, \dots, N\}, \quad (\text{A.4})$$

$$\sum_{k=1}^N \|x_k - x_{k-1}\| \leq \frac{1 + \epsilon_v}{2 + \epsilon_v} 2T. \quad (\text{A.5})$$

Moreover, as ϵ_x decreases to zero, the integer N remains uniformly bounded.

Without loss of generality, it is assumed that $\epsilon_\delta > \epsilon_x$.

In what follows, given a path $\rho : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} and a time $t \in [0, T]$, we use the notation $\rho(t^+)$ and $\rho(t^-)$ to denote the limits of $\rho(s)$ as $s \rightarrow t$ from above and below, respectively.

Proof: Let $\bar{\mathcal{R}} := \{\mathcal{R}_1, \dots, \mathcal{R}_K\}$ be a convex partition of \mathcal{R} such that its diameter d_i satisfies $d_i \leq \gamma \epsilon_\delta$ for each $i \in \{1, 2, \dots, K\}$, where $\gamma \in (0, 1)$ is a constant that will be specified shortly. Next, let \mathcal{X} be a finite set of points on the boundaries of $\bar{\mathcal{R}}$, which we denote by $\partial \bar{\mathcal{R}}$, that is dense enough so that for every point $r \in \partial \bar{\mathcal{R}}$, there exists a point $x \in \mathcal{X}$ for which $\|r - x\| \leq \alpha \epsilon_x$, where $\alpha \in (0, 1)$ is another constant that we will specify shortly.

We now construct a sequence $\{\tau\}$ of times starting from $\tau_0 := 0$ and a sequence $\{x\}$ of points in $\bar{\mathcal{X}}$ starting from $x_0 = \rho(0)$ that satisfy (A.1)-(A.5). Each τ_k corresponds to the next instant in time greater than or equal to $\tau_{k-1} + t_{\min}$ that $\rho(t)$ crosses the boundary of the convex partition, where t_{\min} is an arbitrary

positive time. Formally, we express this as $\tau_k := \min\{t : t > \tau_{k-1} + t_{\min} \text{ and } \rho(t) \in \partial\bar{\mathcal{R}}\}$.

We then choose x_k to be the point in \mathcal{X} nearest to the point $\rho(\tau_k)$, *i.e.* $x_k := \arg \min_{x \in \mathcal{X}} \|x - \rho(\tau_k)\|$. Since there exists a point x such that $\|r - x\| \leq \alpha\epsilon_x$ for each point r on the boundary $\partial\mathcal{R}_i$, we conclude that (A.3) holds. Using the fact that $\|\dot{\rho}(t)\| = 1$, we can bound the maximum distance between two points in the sequence by

$$\|x_k - x_{k-1}\| \leq d_i + t_{\min} \leq \gamma\epsilon_\delta + t_{\min}.$$

We now require that $\gamma < \frac{\epsilon_\delta - t_{\min}}{\epsilon_\delta}$, which allows us to conclude that $\|x_k - x_{k-1}\| \leq \epsilon_\delta$ and therefore (A.1) holds. Also, since $\rho(t)$ is guaranteed to lie inside a single element \mathcal{R}_i of the partition on the the time interval $[\tau_{k-1} + t_{\min}, \tau_k]$, it follows that (A.2) holds.

To prove (A.4), we define $\delta_k := \dot{\rho}(\tau) - \frac{x_k - x_{k-1}}{\tau_k - \tau_{k-1}}$, $\tau \in (\tau_{k-1}, \tau_k)$. Since

$$x_k - x_{k-1} = (\tau_k - \tau_{k-1})(\dot{\rho}(\tau) - \delta_k), \tag{A.6}$$

we conclude that

$$\begin{aligned} \left\| \dot{\rho}(\tau) - \frac{x_k - x_{k-1}}{\tau_k - \tau_{k-1}} \right\| &= \left\| \dot{\rho}(\tau) - \frac{\dot{\rho}(\tau) - \delta_k}{\|\dot{\rho}(\tau) - \delta_k\|} \right\| \\ &= \left\| \left(1 - \frac{1}{\|\dot{\rho}(\tau) - \delta_k\|}\right) \dot{\rho}(\tau) + \frac{\delta_k}{\|\dot{\rho}(\tau) - \delta_k\|} \right\| \\ &\leq \frac{|\|\dot{\rho}(\tau) - \delta_k\| - 1|}{\|\dot{\rho}(\tau) - \delta_k\|} + \frac{\|\delta_k\|}{\|\dot{\rho}(\tau) - \delta_k\|} \leq \frac{2\|\delta_k\|}{1 - \|\delta_k\|}. \end{aligned}$$

Thus (A.4) will hold, provided that

$$\frac{2\|\delta_k\|}{1 - \|\delta_k\|} \leq \epsilon_v \Leftrightarrow \|\delta_k\| \leq \frac{\epsilon_v}{2 + \epsilon_v}.$$

One can show that the duration of each time interval $[\tau_{k-1}, \tau_k]$ is bounded from above by a class \mathcal{K} function of the maximum diameter of a region in the partition, provided that the path cannot complete a loop inside an element of the partition. Formally, we say that $\tau_k - \tau_{k-1} \leq \theta(\gamma\epsilon_\delta)$, where $\theta(\cdot)$ is a class \mathcal{K} function, under the requirement that γ is chosen to satisfy $\gamma\epsilon_\delta < \frac{1}{A}$, recalling that $\|\ddot{\rho}(t)\| \leq A$. Using this fact along with the Mean Value Theorem, we conclude that

$$\begin{aligned} \|\delta_k\| &= \left\| \dot{\rho}(\tau) - \frac{x_k - \rho(\tau_k) - x_{k-1} + \rho(\tau_{k-1}) + \rho(\tau_k) - \rho(\tau_{k-1})}{\tau_k - \tau_{k-1}} \right\| \\ &\leq \frac{\|x_k - \rho(\tau_k)\| + \|x_{k-1} - \rho(\tau_{k-1})\|}{\tau_k - \tau_{k-1}} + \left\| \dot{\rho}(\tau) - \frac{\rho(\tau_k) - \rho(\tau_{k-1})}{\tau_k - \tau_{k-1}} \right\| \\ &\leq \frac{2\alpha\epsilon_x}{\tau_k - \tau_{k-1}} + \frac{A}{2}(\tau_k - \tau_{k-1}) \leq \frac{2\alpha\epsilon_x}{t_{\min}} + \frac{A\theta(\gamma\epsilon_\delta)}{2}. \end{aligned}$$

By selecting sufficiently small α and γ , one can ensure that $\|\delta_k\| \leq \frac{\epsilon_v}{2 + \epsilon_v}$, and therefore (A.4) holds. The consequences of this are that a small value for α will necessarily require a fine sampling of $\partial\bar{\mathcal{R}}$, and a small value for γ will require smaller diameters d_i and thus more partition elements $\bar{\mathcal{R}}$.

Finally, from (A.6) and the fact that $\|\dot{\rho}(t)\| = 1$, we conclude that

$$\|x_k - x_{k-1}\| \leq (\tau_k - \tau_{k-1})(1 + \|\delta_k\|) \leq \frac{2(1 + \epsilon_v)}{2 + \epsilon_v}(\tau_k - \tau_{k-1}),$$

from which (A.5) follows. Note that as one decreases ϵ_x , we need a finer sampling of $\partial\bar{\mathcal{R}}$ but t_{\min} need not decrease and therefore N remains bounded. \square

Proposition 1 *Given a path $\rho : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} and an interval $(t_1, t_2) \subset [0, T]$ on which ρ is twice continuously differentiable,*

$$|\ell(\rho(t_2), \dot{\rho}(t_2^+)) - \ell(\rho(t_1), \dot{\rho}(t_1^-))| \leq g_x(t_2 - t_1) + a g_v(t_2 - t_1), \quad (\text{A.7})$$

where

$$g_x := \sup_{t \in (t_1, t_2)} \|\nabla_x \ell(\rho(t), \dot{\rho}(t))\|, \quad g_v := \sup_{t \in (t_1, t_2)} \|\nabla_v \ell(\rho(t), \dot{\rho}(t))\|, \quad a := \sup_{t \in (t_1, t_2)} \|\ddot{\rho}(t)\|.$$

Proof: Since ℓ is continuously differentiable on (t_1, t_2) , by the Mean Value Theorem we conclude that

$$|\ell(\rho(t_2), \dot{\rho}(t_2^+)) - \ell(\rho(t_1), \dot{\rho}(t_1^-))| \leq g_x \|\rho(t_2) - \rho(t_1)\| + g_v \|\dot{\rho}(t_1^+) - \dot{\rho}(t_2^-)\|.$$

Since $\|\dot{\rho}\| \leq 1$ on (t_1, t_2) , we conclude that $\|\rho(t_2) - \rho(t_1)\| \leq t_2 - t_1$ because of the Mean Value Theorem. Similarly, we can use the Mean Value Theorem to show that $\|\dot{\rho}(t_1^+) - \dot{\rho}(t_2^-)\| \leq a(t_2 - t_1)$ and therefore (A.7) holds. \square

Lemma 2 *Given two paths $\rho_i : [0, T] \rightarrow \mathcal{R}$ in \mathcal{P} , for $i \in \{1, 2\}$ and an interval $(t_1, t_2) \subset [0, T]$ on which each ρ_i is twice continuously differentiable and lies entirely inside the region $\tilde{\mathcal{R}}_j \subset \mathcal{R}$, and a constant $\delta > 0$,*

$$\int_{t_1}^{t_2} \ell(\rho_2(t), \dot{\rho}_2(t)) dt - \int_{t_1}^{t_2} \ell(\rho_1(t), \dot{\rho}_1(t)) dt \leq g_x \|\rho_2(t_1) - \rho_1(t_1)\| \Delta + g_v \|\dot{\rho}_2(t_1^+) - \dot{\rho}_1(t_1^+)\| \Delta + (g_x + \frac{a_1 + a_2}{2} g_v) \Delta^2 \quad (\text{A.8})$$

where

$$g_x := \sup_{x \in \tilde{\mathcal{R}}_j, v \in \mathcal{V}} \|\nabla_x \ell(x, v)\|, \quad \Delta := \min \{ \|\rho_1(t_2) - \rho_1(t_1)\|, \|\rho_2(t_2) - \rho_2(t_1)\| \},$$

$$g_v := \sup_{x \in \tilde{\mathcal{R}}_j, v \in \mathcal{V}} \|\nabla_v \ell(x, v)\|, \quad a_i := \sup_{t \in (t_1, t_2)} \|\ddot{\rho}_i(t)\|,$$

$$\tilde{\mathcal{R}}_j := \mathcal{R}_j \cup \bigcup_{r \in \mathcal{R}_j} B_\delta(r),$$

and $B_\delta(r)$ denotes the closed ball of radius δ centered at the point r .

Proof: For simplicity of notation let us define

$$\ell_i(t) := \begin{cases} \ell(\rho_i(t), \dot{\rho}_i(t)) & t \in (t_1, t_2) \\ \ell(\rho_i(t), \dot{\rho}_i(t^+)) & t = t_1 \\ \ell(\rho_i(t), \dot{\rho}_i(t^-)) & t = t_2 \end{cases} \quad i \in \{1, 2\}.$$

From the fact that $\|\dot{\rho}_i\| = 1$, the Mean-Value Theorem, and the definition of Δ we conclude that

$$t_2 - t_1 \geq \|\rho_i(t_2) - \rho_i(t_1)\| \geq \Delta. \quad (\text{A.9})$$

Using the triangle inequality, we can bound the cost difference at each time $t \in (t_1, t_2)$ by the sum of the following three terms:

$$|\ell_2(t) - \ell_1(t)| \leq |\ell_2(t) - \ell_2(t_1)| + |\ell_1(t) - \ell_1(t_1)| + |\ell_2(t_1) - \ell_1(t_1)|.$$

From Proposition 1, we conclude that

$$|\ell_i(t) - \ell_i(t_1)| \leq (g_{x,i} + a_i g_{v,i})(t_2 - t_1) \quad \forall i \in \{1, 2\},$$

where

$$g_{x,i} := \sup_{t \in (t_1, t_2)} \|\nabla_x \ell(\rho_i(t), \dot{\rho}_i(t))\|, \quad g_{v,i} := \sup_{t \in (t_1, t_2)} \|\nabla_v \ell(\rho_i(t), \dot{\rho}_i(t))\|.$$

Using the fact that each $g_{x,i}$ and $g_{v,i}$ is bounded above by g_x and g_v , respectively (because the paths ρ_i lie entirely inside the region $\tilde{\mathcal{R}}_j$), we combine the expressions to arrive at

$$|\ell_2(t) - \ell_1(t)| \leq (2g_x + (a_1 + a_2)g_v)(t_2 - t_1) + |\ell_2(t_1) - \ell_1(t_1)|.$$

We can generate an upper bound on the last term by using the Mean Value Theorem:

$$|\ell_2(t_1) - \ell_1(t_1)| \leq g_x \|\rho_2(t_1) - \rho_1(t_1)\| + g_v \|\dot{\rho}_2(t_1^+) - \dot{\rho}_1(t_1^+)\|.$$

We now have

$$|\ell_2(t) - \ell_1(t)| \leq (2g_x + (a_1 + a_2)g_v)(t_2 - t_1) + g_x \|\rho_2(t_1) - \rho_1(t_1)\| + g_v \|\dot{\rho}_2(t_1^+) - \dot{\rho}_1(t_1^+)\|.$$

Integrating this expression on the interval (t_1, t_2) and applying (A.9) results in (A.8) and the proof is completed. \square

Bibliography

- [1] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2), 2002.
- [2] Stephan Bohacek, João Pedro Hespanha, and Katia Obraczka. Saddle policies for secure routing in communication networks. volume 2, pages 1416–1421, December 2002.
- [3] Stephen P. Boyd and Craig H. Barratt. *Linear Controller Design: Limits of Performance*. Prentice-Hall, New Jersey, 1991.
- [4] Zhiqiang Chen, Andrew T. Holle, Bernard M. E. Moret, Jared Saia, and Ali Boroujerdi. Network routing models applied to aircraft routing problems. In *Winter Simulation Conference*, pages 1200–1206, 1995.
- [5] B. DasGupta, J. Hespanha, J. Riehl, and E. Sontag. Honey-pot constrained searching with local sensory information. *Nonlinear Analysis: Hybrid Systems and Applications*, 65(9):1773–1793, Nov. 2006.
- [6] J. Eagle and J. Yee. An optimal branch-and-bound procedure for the con-

- strained path, moving target search problem. *Operations Research*, 28(1), 1990.
- [7] Dale Enns, Dan Bugajski, and Steve Pratt. Guidance and control for cooperative search. In *Proceedings of the American Control Conference*, pages 1923–1929, 2002.
- [8] Per-Olof Fjällström. *Algorithms for Graph Partitioning: A Survey*. Linköping University Electronic Press, Linköping, Sweden, 1998.
- [9] J. R. Frost and L. D. Stone. Review of search theory: Advances and applications to search and rescue decision support. Technical report, U.S. Coast Guard Research and Development Center, Gronton, CT, Sept. 2001.
- [10] I. M. Gelfand and S. V. Fomin. *Calculus of Variations*. Selected Publications in the Mathematical Sciences. Prentice-Hall, Englewood Cliffs, N. J., 1963.
- [11] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.
- [12] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM J. Appl. Math*, 9:551–570, 1961.
- [13] K. B. Haley and L. D. Stone, editors. *Search Theory and Applications*. Plenum Press, New York.
- [14] J. P. Hespanha. An efficient matlab algorithm for graph partitioning. Technical report, University of California, Santa Barbara, CA, Oct. 2004. Available at <http://www.ece.ucsb.edu/~hespanha/techreps.html>.

- [15] J. P. Hespanha, H. J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. volume 3, pages 2432–2437, December 1999.
- [16] João P. Hespanha and Huseyin Kızılocak. Efficient computation of dynamic probabilistic maps. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.
- [17] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Comput.*, 22:1199–1217, 1993.
- [18] J. Kim and J. Hespanha. Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of uavs. In *Proc. of the 42nd IEEE Conference on Decision and Control*, 2003.
- [19] B. O. Koopman. *Search and Screening*. Operations Evaluations Group Report No. 56, Center for Naval Analyses, Alexandria, VA, 1946.
- [20] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Shortest anisotropic paths on terrains. In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Proc. 26th Int. Colloquium Automata, Languages and Programming (ICALP'99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 524–533. Springer-Verlag, Berlin, 1999.
- [21] C. Lim, S. Bohacek, J. Hespanham, and K. Obraczka. Hierarchical max-flow routing. In *Proc. of the IEEE GLOBECOM*, 2005.
- [22] M. Mangel. Search theory: A differential equations approach. In D. V. Chudnovsky and G. V. Chudnovsky, editors, *Search Theory: Some Recent Developments*, pages 55–101. Marcel Dekker, New York, 1989.

- [23] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, New York, 1992.
- [24] M. M. Polycarpou, Y. Yang, and K. M. Passino. A cooperative search framework for distributed agents. In *Proc. of the IEEE International Symposium on Intelligent Control*, 2001.
- [25] H. E. Romeijn and R. L. Smith. Parallel algorithms for solving aggregated shortest path problems. *Computers and Operations Research*, 26:941–953, 1999.
- [26] Neil C. Rowe and Ron S. Ross. Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects. *IEEE Trans. Robot. Automat.*, 6(5):540–553, October 1990.
- [27] G. Shen and P. E. Caines. Hierarchically accelerated dynamic programming for finite-state machines. *IEEE Transactions on Automatic Control*, 47(2):271–283, 2002.
- [28] L. D. Stone. *Theory of Optimal Search*. Academic Press, New York, 1975.
- [29] Timothy H. Chung and Joel W. Burdick. A Decision-Making Framework for Control Strategies in Probabilistic Search. In *Intl. Conference on Robotics and Automation*. ICRA, April 2007.
- [30] K. E. Trummel and J.R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.