

Graph Optimization Using Fractal Decomposition With Application to Cooperative Routing Problems

James R. Riehl João P. Hespanha

Center for Control, Dynamical Systems, and Computation
Department of Electrical and Computer Engineering
University of California, Santa Barbara
jriehl@ece.ucsb.edu hespanha@ece.ucsb.edu

Abstract

We introduce a method of hierarchically decomposing graph optimization problems to obtain approximate solutions with low computation. The method uses a partition on the graph to convert the original problem to a high level problem and several lower level problems. On each level, the resulting problems are in exactly the same form as the original one, so they can be further decomposed. In this way, the problems become fractal in nature. We use best-case and worst-case instances of the decomposed problems to establish upper and lower bounds on the optimal criteria, and these bounds are achieved with significantly less computation than what is required to solve the original problem. We show that as the number of hierarchical levels increases, the computational complexity approaches $O(n)$ at the expense of looser approximation bounds. For regular lattice graphs, we provide constant factor bounds on the approximation error. 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.

1 Introduction

Graph optimization problems such as shortest path, maximum flow, and search are essential to a large number of engineering applications including navigation [9], path planning [5], and network routing [12], but for graphs with many nodes, the computation required to solve these problems can be impractical. When the problem calls for cooperation between multiple agents over a network, the complexity grows even more. In these situations when computation of an exact solution would take too much time, it is useful to find fast methods of approximating the solution. This paper introduces a framework of bounding the optimal solution above and below by partitioning the graph and generating best-case and worst-case solutions on a new smaller graph, whose vertices are subsets of the original vertex set. Furthermore, one can use the worst-case solution to generate an approximate solution on the original graph, and the best-case solution provides a bound on how far this approximation is from optimal.

The worst-case solution generally requires the solution of several problems of smaller dimension. These smaller problems are in exactly the same form as the original one so they can be further decomposed, giving the algorithm a recursive hierarchical structure, which we describe as fractal.

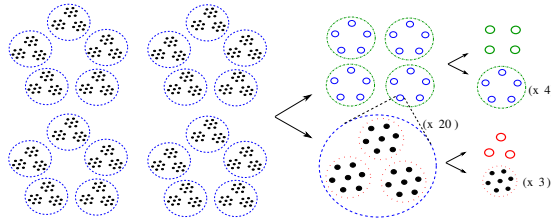


Figure 1: Example of recursive decomposition on a nicely structured graph (edges not shown).

Figure 1 is a diagram of the recursive 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 and generating an approximate solution. We constructed the graph in Figure 1 specifically to illustrate the recursive decomposition process, but the method works on any graph. Increasing the number of decomposition levels reduces computation, but generally results in looser bounds. These bounds depend on the specific instance of the problem, but we show that constant factor approximation bounds do exist for certain regular graphs such as lattices.

The main goal of this paper is to introduce a methodology for recursive hierarchical decomposition of graph optimization problems and to implement it on three well-known problems related to cooperative routing. We will show that this fractal decomposition algorithm greatly reduces computation, and as the number of decomposition levels increases, the computational complexity approaches $O(n)$. Furthermore, for each of the three examples, we provide numerical simulations to demonstrate that the approximation can be quite accurate. First, we introduce the three problems along with previous computational complexity results.

- **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 optimal route planning for groups of UAVs [11]. 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 $O(nm + n^2 \log n)$ [10].

- **Maximum flow matrix.** The maximum flow matrix problem involves finding, for each pair of vertices in a capacitated graph, the flow assignment on the edges that yields the maximum flow intensity. Applications of this problem include stochastic network routing [1] and vehicle routing [2]. Given a directed graph $G(V, E)$ with n vertices and m capacitated edges, Goldberg and Tarjan [6] showed that one can compute the maximum flow between two vertices in $O(nm \log \frac{n^2}{m})$ time. To compute the max-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 [7], but since we are considering 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 max-flow matrix.
- **Cooperative graph search.** The objective of the cooperative graph search problem is to find optimal routes in a graph that maximize the probability that a team of cooperating agents will find a hidden target, subject to a cost constraint on the paths. The computational complexity of the search problem for a single agent is known to be NP-Hard [18] on the number of vertices n . Reducing the s -agent cooperative search problem to a single-agent search problem with n^s vertices results in a problem that is clearly also NP-Hard. In the worst case, an exhaustive search on a complete graph would have complexity $O(n^s!)$. Although there are some more efficient algorithms to solve this problem such as the branch and bound methods of Eagle and Yee [4], this problem is still computationally infeasible for large values of n and s .

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 solve an aggregated all-pairs shortest path problem motivated by minimizing vehicle travel time [15]. 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. The results of our shortest path decomposition example will closely resemble that of [15] with the addition of both upper and lower bounds on the costs of the shortest paths. Also related, Shen and Caines presented results on hierarchically accelerated dynamic programming [16]. 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 bounds.

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 [12]. 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.

DasGupta *et al.* presented an approximate solution for the stationary target search based on an aggregation of the search space using a graph partition [3]. We used a similar approach in [13] while also allowing the partitioning process to be implemented on multiple levels. In [14], we extended the method to cover multiple-agent cooperative search problems.

The remainder of this paper is organized as follows. Section 2 presents the ideas of graph partitions and meta-graphs, introducing concepts and notation that will be used throughout the paper. Section 3 gives an abstract overview of the fractal decomposition methodology, with generic computation results presented in section 4. Sections 5, 6, and 7 define three example problems and give procedures to construct upper and lower bounds on their respective optimal criteria. These sections also include explicit procedures for constructing the approximate shortest path matrix, maximum flow matrix, and cooperative search paths. Next, we give some results on constant factor approximation bounds for lattice graphs. We also include brief numerical examples for shortest path and maximum flow, and a more extensive simulation study on the cooperative search problem. The final section is a discussion of the results with suggestions for future research.

2 Graphs, Meta-graphs, and Subgraphs

This section introduces some notation and terminology that we will use in the remainder of this paper. Given a graph $G := (V, E)$ with vertex set V and edge set $E \subset V \times V$, 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 subgraph $G|\bar{v}_i := (\bar{v}_i, E \cap \bar{v}_i \times \bar{v}_i)$.

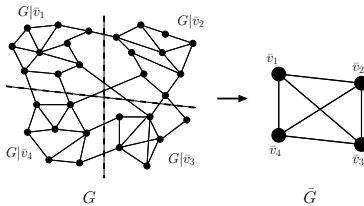


Figure 2: Example of a graph partitioned into 4 *meta-vertices*.

Figure 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 in \tilde{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} .

3 Fractal Decomposition Method

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

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

3.1 Partition the graph

Although the algorithm will work for any partition, some partitions will result in tighter bounds than others, and some will reduce computation more than others. These factors will depend on the specific problem and should be taken into account in the choice of partitioning algorithm. 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-bandwidth edges. The partitioning algorithm we use for the examples in this paper is the one from [8], 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.

3.2 Construct bounding meta-problems and solve

We now want 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-vertex set and meta-edge set defined by the partition. The meta-problems should be in the exact same form as the original problem, so that we may apply any algorithm that solves the original problem to the meta-problems. This property allows for recursive decomposition. The most important step in the construction is assigning data (costs, rewards, bandwidths, *etc.*) to the meta-vertices and meta-edges such that the solutions to the meta-problems are guaranteed to bound the optimal criteria of the original problem. For the best case meta-problem, this step means assigning

the most optimistic values, assuming that we only have data for the meta-graph and not the original graph when solving the meta-problem. Similarly, for the worst-case meta-problem, this step involves assigning pessimistic values to the meta-edges and meta-vertices. The values for each meta-vertex generally come from solving a smaller problem, defined on the corresponding subgraph. 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 will tell us how far our approximate solution is from optimal.

3.3 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. The approximate solution is generated by a refinement process on the worst-case meta-graph, using the solutions to these subproblems, to generate a feasible solution on the original graph. 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 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

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

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

$$\begin{aligned} f(n, k_0) &= f(k_0, k_1) + k_0 f\left(\frac{n}{k_0}, k_2\right) \\ &= f(k_1, 1) + k_1 f\left(\frac{k_0}{k_1}, 1\right) \\ &\quad + k_0 f(k_2, 1) + k_0 k_2 f\left(\frac{n}{k_0 k_2}, 1\right). \end{aligned} \quad (2)$$

A choice of $k = \sqrt{n}$ in (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 gives the following computational complexity results:

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

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 1 shows the computational reduction for various levels of decomposition on the three problems presented in this paper.

Table 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	$O(n^3)$	$O(n^5)$	$O(n^s!)$
1	$O(n^2)$	$O(n^3)$	$O(n^{\frac{1}{2}}(n^{\frac{s}{2}})!)$
2	$O(n^{\frac{3}{2}})$	$O(n^2)$	$O(n^{\frac{3}{4}}(n^{\frac{s}{4}})!)$
3	$O(n^{\frac{5}{4}})$	$O(n^{\frac{3}{2}})$	$O(n^{\frac{7}{8}}(n^{\frac{s}{8}})!)$

In the cooperative search column, s 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.

5 Shortest Path Matrix Problem

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.

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

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

The *path-cost* is given by

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

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 and is denoted 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

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

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 cost and vertex cost defined by

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

Theorem 1 For every partition \bar{V} of G and for every pair of vertices v_{init} and v_{final} ,

$$J_{\bar{G}_{\text{best}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \leq J_G^*(v_{\text{init}}, v_{\text{final}}) \leq J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \quad (3)$$

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

$$\|\bar{G}_{\text{best}}\|_D \leq \|G\|_D \leq \|\bar{G}_{\text{worst}}\|_D. \quad (4)$$

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 detailed 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} be the approximate shortest path in G being constructed. To begin, we want \tilde{p} to include the minimum cost edge of each meta-edge in \bar{p}_w^* . Call this set of edges 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_w^* associated with the meta-edge that connects \bar{v}_j to \bar{v}_{j+1} for $j = (1, 2, \dots, \bar{f} - 1)$, and let $v_{j_{\text{entry}}}$ be the entry vertex of each \bar{v}_j for $j = (2, 3, \dots, \bar{f})$. The incomplete path is then

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

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 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^* :

$$C(\bar{p}_w^*) = \sum_{j=1}^{\bar{f}-1} \bar{c}_{e_{\text{worst}}}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{f}} \|G|\bar{v}_j\|_D \quad (5)$$

$$C(\tilde{p}) = \sum_{j=1}^{\bar{f}-1} \bar{c}_{e_{\text{worst}}}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{f}} C(\tilde{p}_j), \quad (6)$$

where \bar{f} is the number of vertices in the path \tilde{p} . Because we have defined each \tilde{p}_j to be a 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 (6) must be less than or equal to the last term of (5). Since the edge cost terms are identical, we conclude that

$$C(p^*) \leq C(\bar{p}) \leq C(\bar{p}_w^*), \quad (7)$$

where p^* is the shortest path in G from v_{init} to v_{final} . The left inequality in (7) holds because of the optimality of p^* , and the right inequality was discussed above. Therefore, the upper bound in (3) 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 check 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 $\bar{p}_b = (\bar{v}_1, \bar{v}_2, \dots, \bar{v}_{\bar{f}})$, where \bar{f} 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 $j \in [1, \bar{f}]$. The remaining edges connecting consecutive \bar{v}_j along p^* are contained in the set difference $\bar{E}_{p_j^*} := E_{p^*} \setminus \hat{E}_{p_j^*}$.

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

$$C(\bar{p}_{\text{best}}) = \sum_{j=1}^{\bar{f}-1} \bar{c}_{e_b}(\bar{v}_j, \bar{v}_{j+1}) + \sum_{j=1}^{\bar{f}} \bar{c}_{v_{\text{best}}}(\bar{v}_j) \quad (8)$$

$$C(p^*) = \sum_{e \in \bar{E}_{p^*}} c_e(e) + \sum_{e \in \hat{E}_{p^*}} c_e(e) + \sum_{i=1}^f c_v(v_i), \quad (9)$$

where f is the length of p^* .

The first term on the right side of (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 (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 (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 (9), we see that the lower bound in 3 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

5.1 Error Bounds for Regular 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. While this is quite a difficult problem for arbitrary graphs, we can generate constant factor approximation bounds for the diameters of some regular graphs, such as lattices.

To begin, let us consider the shortest-path problem on two-dimensional square lattice graphs $G(n) := L^{\sqrt{n} \times \sqrt{n}}$. 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 $G(n)$ 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

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

If we again use the fractal decomposition algorithm to approximate the sub-graph diameters, which generate the meta-vertex costs of $\bar{G}_{\text{worst}}(n)$, 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 lattice graphs.

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

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

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

Proof: The diameter of $G(n)$ 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

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

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

5.2 Case Study on Diameter Approximation

This section presents some results of the fractal decomposition approximation to the diameters of two test graphs: a Delaunay graph with clustered vertices, and

a lattice graph. One would expect a better approximation for the first graph than the second because in the clustered graph, we can find a partition that generates a meta-graph in which the meta-edges have much higher cost than the edges within subgraphs, while this is not the case in the lattice graph.

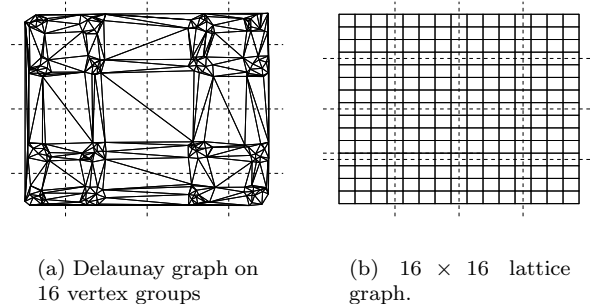


Figure 3: The graph on the left was created by randomly distributing vertices over 16 $1 \text{ unit} \times 1 \text{ unit}$ regions centered in a block pattern and generating a Delaunay graph over this vertex set. The graph on the right is a two-dimensional rectangular lattice graph. Dashed lines indicate the partitions.

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 in the meta-graph. Table 2 shows the results for best-case, worst-case, approximate, and actual diameters for each test graph partitioned into 16 vertex groups. The approximate diameter is computed using the procedure outlined in the proof of Theorem 1, and will always lie between the actual and worst-case diameters.

Table 2: Results of Diameter Approximation for Test

	Best-case	Actual	Approximate	Worst-case
Grouped	9.0	14.1	14.5	15.1
Lattice	5	30	30	48

As expected, the worst-case bounds are fairly tight for the clustered graph, but worse for the lattice graph. The worst-case meta-graph diameter of 48 lies within the constant factor bound of 2 derived in section 5.1. Although the approximation generates the correct diameter for the lattice graph, there is a large uncertainty due to the best-case lower bound. We conclude that using fractal decomposition to approximate the shortest path matrix problem works best on graphs with some inherent clustered structure.

6 Maximum Flow Matrix Problem

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 bandwidth or capacity of that edge. In this formulation, the flow through each vertex is also limited by a bandwidth. This vertex bandwidth is what allows for the hierarchical decomposition of this problem.

Data

$G := (V, E)$	directed graph
$b_e : E \rightarrow [0, \infty)$	edge bandwidth function
$b_v : V \times O \rightarrow [0, \infty)$	vertex bandwidth 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 exist some $\mu \geq 0$ such that

$$f_{\text{out}}(v) - f_{\text{in}}(v) = \begin{cases} \mu & v = v_{\text{init}} \\ -\mu & v = v_{\text{final}} \\ 0 & \text{otherwise,} \end{cases} \quad \forall v \in V, \quad (10)$$

$$0 \leq f(e) \leq b_e(e), \quad \forall e \in E \quad (11)$$

$$0 \leq f_{\text{in}}(v) \leq \nu(v), \quad \forall v \in V \quad (12)$$

$$0 \leq f_{\text{out}}(v) \leq \nu(v), \quad \forall v \in V \quad (13)$$

In the above,

$$f_{\text{in}}(v) := \sum_{e \in \text{In}[v]} f(e), \quad f_{\text{out}}(v) := \sum_{e \in \text{Out}[v]} f(e), \quad (14)$$

where $\text{In}[v] \in E$ denotes the set of edges that enter the vertex v and $\text{Out}[v] \in E$ denotes the set of edges that exit 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 *bandwidth* of the graph and is denoted by $\|G\|_B$.

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

$$\bar{b}_{e_{\text{worst}}}(\bar{e}) := \sum_{e \in \bar{e}} b_e(e) \quad \bar{b}_{v_{\text{worst}}}(\bar{v}) := \|G|\bar{v}\|_B. \quad (15)$$

Note that to construct the graph \tilde{G}_{worst} one needs to compute the bandwidth 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 bandwidth and vertex bandwidth defined by

$$\bar{b}_{e_{\text{best}}}(\bar{e}) := \sum_{e \in \bar{e}} b_e(e) \quad \bar{b}_{v_{\text{best}}}(\bar{v}) := \sum_{v \in \bar{v}} b_v(v). \quad (16)$$

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

$$J_{\bar{G}_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \leq J_G^*(v_{\text{init}}, v_{\text{final}}) \leq J_{\bar{G}_{\text{best}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}) \quad (17)$$

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

$$\|\bar{G}_{\text{worst}}\|_B \leq \|G\|_B \leq \|\bar{G}_{\text{best}}\|_B. \quad (18)$$

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 $\bar{f}_{\text{worst}}^*(\bar{e})$ in \bar{G}_{worst} , that satisfies (10)–(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 bandwidth 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} , out of which we now construct a flow $\tilde{f}(e)$ in G that satisfies (10)–(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:

$$\bar{f}_w^*(\bar{e}) = \tilde{f}(\bar{e}_1) + \tilde{f}(\bar{e}_2) + \cdots + \tilde{f}(\bar{e}_p), \quad (19)$$

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 (11) holds for the worst-case meta-graph, that is

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

we know that a decomposition (19) exists whose flows satisfy condition (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

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

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:

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

where \tilde{f}_{in} and \tilde{f}_{out} are as defined in (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. Since each subgraph $G|\bar{v}$ is connected, 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 (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 bandwidth of the subgraph $G|\bar{v}$, the flows generated by this procedure will satisfy (11)-(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,

$$J_G^*(v_{\text{init}}, v_{\text{final}}) \geq J_{G_{\text{worst}}}^*(\bar{v}_{\text{init}}, \bar{v}_{\text{final}}). \quad (20)$$

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

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 (10) and (11), it follows from (16) and the above equation that $\hat{f}_{\text{best}}(\bar{e})$ satisfies (10) and (11). Since $\bar{b}_{v_{\text{best}}}(\bar{v}) = \sum_{v \in \bar{v}} b_v(v)$, (12) and (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 $\|G\|_B \leq \|\bar{G}_{\text{best}}\|_B$. \square

6.1 Error Bounds for Regular 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 bandwidths are one and all vertex bandwidths 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 2 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 bandwidth 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 bandwidths equal to 2 and all meta-vertex bandwidths equal to 2. The bandwidth of this meta-graph is 2, the same as the bandwidth of the original graph. We now generalize this result to d -dimensional lattices.

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

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

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 bandwidths equal to $n^{\frac{d-1}{2d}}$ and all meta-vertex bandwidths 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 bandwidth of a 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.

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

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

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 2 with the addition of the toroidal meta-edges. However, since the meta-vertex bandwidths do not change, the bandwidth of the meta-graph remains equal to d . \square

6.2 Case Study on Bandwidth Approximation

To test the fractal decomposition algorithm on the maximum flow matrix problem, we used one of the graphs from [12], which was generated from the Verio Internet service provider (ISP) topology computed in [17]. Game theoretic stochastic routing (GTSR) was introduced by Bohacek *et al.* [1] to increase robustness of network routing. Suppose that in the event of a fault on a communication link l (represented by a graph edge), a percentage p_l of packets are dropped. The main result of GTSR is that the routing policy that minimizes packet drop under these conditions is the one given by solving a maximum flow problem on a graph in which the edge bandwidth corresponding to each link l is $\frac{1}{p_l}$.

Partitioning the graph in Figure 4(a) into 10 parts and applying the fractal decomposition algorithm described in the previous section yields the results in Table 3. In this case, the fractal decomposition algorithm generates the correct

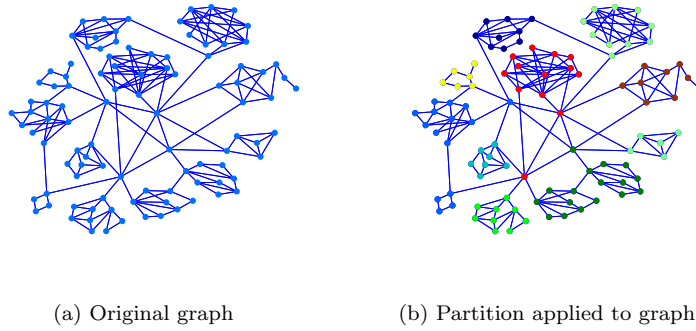


Figure 4: Graph of the Verio ISP topology (left) partitioned into 10 subgraphs (right).

bandwidth of the graph, but it is not necessarily true that the maximum flow approximation between every pair of source and destination vertices is also correct. However, similar to the result in [12], if the meta-vertex bandwidths along the optimal flow in the worst-case meta-graph are not limiting the flow, *i.e.* the constraints corresponding to these meta-vertices are not active, the best-case and worst-case meta-graph diameters will be equal. Generally, this means that if the meta-vertices have high bandwidths compared to the meta-edges, the fractal decomposition algorithm generates the correct bandwidth.

Table 3: Fractal Decomposition Results for the Max-Flow Routing Problem on the Verio ISP Topology

	Bandwidth
Worst-case	0.1432
Actual	0.1432
Best-case	0.1432

7 Cooperative Graph Search Problem

Consider a team of s agents searching for one or more objects 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 searcher that maximize the total reward collected by the team subject to a cost constraint on the individual agents.

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, $o_{\max} \in \{1, 2, \dots, s\}$ 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,

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

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

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

and the *path-reward* is given by

$$R(p) := \sum_{v \in p} r(v), \quad (21)$$

where the sum in (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 $C(p) \leq L$.

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 s 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, *i.e.* the full graph will consist of up to n^s 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 s -length vectors whose entries are the vertex locations in V of each member of the team. We write the expanded team-vertex as $\mathbf{v} = (\mathbf{v}(1), \mathbf{v}(2), \dots, \mathbf{v}(s))$, 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 searchers 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 searchers 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,

$$\mathbf{p} := (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{f-1}, \mathbf{v}_f), \quad (\mathbf{v}_i, \mathbf{v}_{i+1}) \in \mathbf{E}, \quad (22)$$

where f is the length of the path. Let \mathbf{p}_a denote the path of agent a , that is $\mathbf{p}_a := (\mathbf{v}_1(a), \dots, \mathbf{v}_f(a))$. 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

$$\mathbf{R}(\mathbf{p}) := \sum_{i=1}^f \sum_{v \in \mathbf{v}_i} r(v, o_{\mathbf{v}_i}^v) \kappa(v, i), \quad (23)$$

where $o_{\mathbf{v}_i}^v$ is the occupancy of v when the team is at \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 , denote the maximum reward that s searchers can collect on G by $J_G^*(s, L)$. We can write the objective for the cooperative search problem as follows:

$$J_G^*(s, L) := \max_{\mathbf{p}} \mathbf{R}(\mathbf{p}) \quad \text{s. t.} \quad \mathbf{C}(\mathbf{p}) \leq L. \quad (24)$$

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

The meta-vertex cost and reward functions are defined by solving cooperative search problems on their corresponding subgraphs:

$$r_{\text{worst}}(\bar{v}, \bar{o}) := J_{G|\bar{v}}^*(\bar{o}, l), \quad (25)$$

$$c_{v_{\text{worst}}}(\bar{v}, \bar{o}) := \mathbf{C}(\mathbf{p}^*(\bar{v}, \bar{o})), \quad \forall \bar{o} \in \bar{O}, \forall \bar{v} \in \bar{V}, \quad (26)$$

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 5 diagrams this process for two meta-vertices for which $o_{\text{max}} = 2$. 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

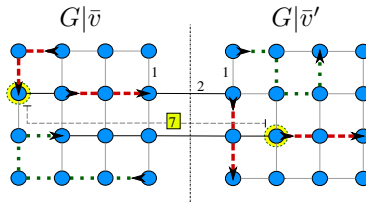


Figure 5: Example showing the worst-case edge-cost between two meta-vertices. Edge-costs are 1 for edges within subgraphs, 2 for edges between subgraphs and all vertex costs are 0.

worst case meta-edge cost function for all $(\bar{v}, \bar{v}') \in \bar{E}$, as

$$c_{e_{\text{worst}}}(\bar{v}, \bar{v}') = \max_{\bar{o}, \bar{o}'} \max_{a, a'} d[p_a^*(\bar{v}, \bar{o}, f), p_{a'}^*(\bar{v}', \bar{o}', 1)], \quad (27)$$

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 of notation, we choose an edge-cost on the location meta-graph that does not depend on the vertex occupancies.

Now let $\bar{\mathbf{G}}_{\text{worst}} := (\bar{\mathbf{V}}, \bar{\mathbf{E}})$ 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 (22) and (23). The objective in the worst-case cooperative meta-graph search problem is to solve the cooperative search problem (24) on \bar{G}_{worst} and thus find the reward $J_{\bar{G}_{\text{worst}}}^*(s, L)$.

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 (24). Let $\bar{G}_{\text{best}} := (\bar{V}, \bar{E})$ be a meta-graph of G with edge cost function defined by

$$c_{e_{\text{best}}}(\bar{v}, \bar{v}') = \min_{v \in \bar{v}, v' \in \bar{v}'} d[v, v'], \quad \forall (\bar{v}, \bar{v}') \in \bar{E}. \quad (28)$$

where $d[v, v']$ is the cost of the shortest path in G from v to v' (for the example in Figure 5, $c_{e_{\text{best}}}(\bar{v}, \bar{v}') = 2$). Now, set $o_{\text{max}} = 1$ and define the meta-vertex reward and cost functions as

$$r_{\text{best}}(\bar{v}, 1) := \sum_{v \in \bar{v}} \max_o r(v, o) \quad (29)$$

$$c_{v_{\text{best}}}(\bar{v}, 1) := \min_{v \in \bar{v}} c_v(v, 1). \quad (30)$$

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 (24) on \bar{G}_{best} and thus find the reward $J_{\bar{G}_{\text{best}}}^*(s, L)$.

Theorem 3 *For every partition \bar{V} of G*

$$J_{\bar{G}_{\text{worst}}}^*(s, L) \leq J_G^*(s, L) \leq J_{\bar{G}_{\text{best}}}^*(s, L) \quad (31)$$

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 $J_{\bar{G}_{\text{worst}}}^*(s, L)$ and $J_G^*(s, L)$.

Proof: To verify the lower bound, we use the optimal worst-case team path $\bar{\mathbf{p}}^* = (\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \dots, \bar{\mathbf{v}}_f)$ 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,

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

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 ,

$$\hat{\mathbf{p}}_a = (\mathbf{p}_a^*(\bar{\mathbf{v}}_1(a), \bar{o}), \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_2(a), \bar{o}), \dots, \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_{f-1}(a), \bar{o}), \dots, \mathbf{p}_a^*(\bar{\mathbf{v}}_f(a), \bar{o})),$$

Because of (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 (25), so we know that $J_{\bar{G}_{\text{worst}}}^*(s, L) = \mathbf{R}(\bar{\mathbf{p}}^*) \leq \mathbf{R}(\hat{\mathbf{p}})$. 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 (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} , $\mathbf{R}(\hat{\mathbf{p}}) \leq J_G^*(s, L)$, and the lower bound in 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 $\mathbf{p}^* = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_f)$ in \mathbf{G} that generates reward $R_G^*(s, L)$. 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 (28) and (30) that $C(\tilde{\mathbf{p}}) \leq C(\mathbf{p}^*)$, so $\tilde{\mathbf{p}}$ meets the cost constraint. Finally, due to (29), all reward is collected from each meta-vertex visited by $\tilde{\mathbf{p}}$, and the upper bound $J_G^*(s, L) \leq J_{\bar{G}_{\text{best}}}^*(s, L)$ holds. \square

7.1 Search Approximation for Regular Graphs

Here we investigate the accuracy of the fractal decomposition algorithm for the search problem on certain regular 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.

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

$$\lim_{n \rightarrow \infty} \frac{J_{\bar{G}_{\text{worst}}(n)}^*(1, \gamma n)}{J_{G(n)}^*(1, \gamma n)} = 1. \quad (32)$$

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. □

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 . This extends recursively for multiple decomposition levels. 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.

7.2 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 6 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.

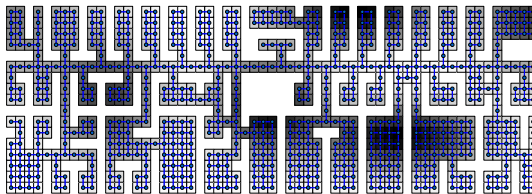
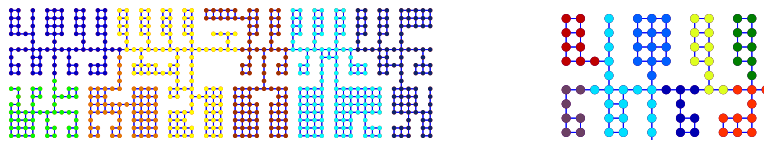


Figure 6: Model of third floor of UCSB’s Harold Frank Hall divided into 646 cells and overlaid with a graph. Dark cells indicate high target probability.

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 get an idea of 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 [8], 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 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 7(a) shows the top-level partition on our test graph, with vertices of the same color belonging to the same partitioned sub-graph. Figure 7(b) shows the second-level partition applied to the subgraph in the upper left corner of Figure 7(a). The remaining subgraphs are similarly partitioned.



(a) Top-level partition on the graph into 7 sub-graphs.

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

Figure 7: Two levels of partitioning on the search graph.

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 8 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,

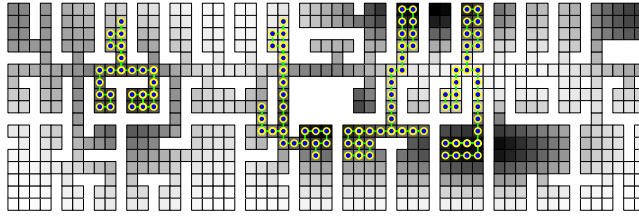


Figure 8: Results of 4-agent cooperative search simulation.

which lies between the worst-case lower bound of 0.26 and best-case upper bound of 1.0. Table 4 shows the results of the algorithm for one to four searchers. The

Table 4: Results of Cooperative Search for 1-4 Agents

Searchers	Cost	Reward	$J_{\mathcal{G}_{\text{worst}}}^*(s, 120)$	$J_{\mathcal{G}_{\text{best}}}^*(s, 120)$
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 s searchers are able to collect almost s 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 issue for future research.

There is also some backtracking along the paths in Figure 8, 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.

8 Conclusions and Future Work

We have introduced a method of decomposing graph optimization problems to achieve upper and lower bounds on the optimal criteria with much less computation than what is required to solve the complete problems. Additionally,

the problems are formulated in such a way that allows for recursive hierarchical decomposition. As the number of levels increases, the computational complexity approaches $O(n)$ at the expense of looser bounds on the optimal criteria. We gave three example problems to demonstrate the implementation of this algorithm: shortest path matrix, maximum flow matrix, and cooperative search. Although we cannot guarantee constant factor approximation bounds for arbitrary graphs, we provide some such results for lattice graphs. For the shortest path matrix problem on d -dimensional lattice graphs, we showed that an L -level decomposition algorithm always approximates the diameter to within d^L of the actual diameter. For the maximum flow problem, the fractal decomposition algorithm generates the correct bandwidth for d -dimensional lattice graphs. If we apply the cooperative search approximation to a lattice graph, the expected reward collected approaches the optimal reward as we increase the size of the lattice.

We also provided numerical case studies for the three example problems. The results of the shortest path matrix tests showed that the algorithm performs best on graphs with clustered vertices. We tested the maximum flow approximation on a graph generated from an actual Internet topology and it yielded the correct bandwidth. We also ran the cooperative search algorithm on a floor model of a large university building, and showed that the fractal decomposition algorithm is computationally fast and achieves true cooperation between agents.

There are several potential directions for future work on the fractal decomposition method. One is to generate distributed versions of the algorithms. Another is to find more potential applications of this approach. For example, in the cooperative search problem, we would like to generalize to the moving target case, a somewhat more difficult problem because the target probability distribution is constantly changing as time passes and new information is collected.

References

- [1] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka. Game theoretic stochastic routing for fault tolerance on computer networks. To appear in the IEEE Transactions on Parallel and Distributed Systems.
- [2] Z. Chen, A. T. Holle, B. M. E. Moret, J. Saia, and A. Boroujerdi. Network routing models applied to aircraft routing problems. In *Winter Simulation Conference*, pages 1200–1206, 1995.
- [3] 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.
- [4] J. Eagle and J. Yee. An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Operations Research*, 28(1), 1990.

- [5] J. A. Fernandez and J. Gonzales. Hierarchical path search for mobile robot path planning. In *Proc. IEEE Int'l Conf. Robotics and Automation*, 1998.
- [6] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.
- [7] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM J. Appl. Math.*, 9:551–570, 1961.
- [8] 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>.
- [9] N. Jing, Y. Huang, and E. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *Proc. Fifth Int'l Conf. Info. and Know. Mgmt.*, pages 261–268, 1996.
- [10] 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.
- [11] 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.
- [12] C. Lim, S. Bohacek, J. Hespanham, and K. Obraczka. Hierarchical max-flow routing. In *Proc. of the IEEE GLOBECOM*, 2005.
- [13] J. R. Riehl and J. P. Hespanha. Fractal graph optimization algorithms. In *Proceedings of the 44th Conference on Decision and Control*, 2005.
- [14] J. R. Riehl and J. P. Hespanha. Cooperative graph search using fractal decomposition. To be presented at the ACC'07, July 2007.
- [15] H. E. Romeijn and R. L. Smith. Parallel algorithms for solving aggregated shortest path problems. *Computers and Operations Research*, 26:941–953, 1999.
- [16] G. Shen and P. E. Caines. Hierarchically accelerated dynamic programming for finite-state machines. *IEEE Transactions on Automatic Control*, 47(2):271–283, 2002.
- [17] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *Transactions on Networking*, 12, 2004.
- [18] K. E. Trummel and J. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.