SHORTEST PATHS IN EUCLIDEAN GRAPHS*

(extended abstract)

Robert Sedgewick
Jeffrey Scott Vitter

Department of Computer Science
Brown University
Providence, R.I. 02912

Abstract. We analyze a simple method for finding shortest paths in *Euclidean graphs* (where vertices are points in a Euclidean space and edge weights are distances between points). For many graph models, the running time of the algorithm to find the shortest path between a specified pair of vertices in a graph with V vertices and E edges is shown to be $O(V)$ as compared with $O(V \log V + E)$ required by the classical (Dijkstra) algorithm.

1. INTRODUCTION

It has long been known that the efficiency of search procedures in graphs can be improved by using global information to evaluate partial solutions and so direct the search (see [Hart, Nilsson, and Raphael, 68]). This technique has mostly been applied to develop heuristics for finding partial or approximate solutions to difficult or intractable search problems, but it can be used to improve elementary search algorithms on important classes of graphs. For the particular problem of finding the shortest path between two graph vertices, we are able to precisely characterize the improvement through average-case analysis for a variety of random graph models.

Specifically, we're interested in *Euclidean graphs*, where each vertex corresponds to a point in a Euclidean space and where the weight of an edge is proportional to the Euclidean distance between the points it connects. Such graphs model many real situations (airline route map, some circuit layout applications, etc.). Typically, the inherent geometric information in such graphs is ignored and the classical traversal algorithms for general graphs are used; in fact it is possible to gain efficiency by making use of the geometric information directly within the graph traversal algorithm.

Euclidean graphs do arise indirectly in the solution of some geometric problems, though it is still typical to separate geometric considerations from graph processing. For example, a standard technique for finding the *minimum spanning tree* of a set of points in Euclidean space is to build a sparse subgraph of the complete graph induced by the points, then use an algorithm for general graphs.

Classical graph traversal algorithms can be developed according to the following general schema: To visit all the vertices of a given graph, place one distinguished *start* vertex in a priority queue. Then repeatedly perform the following operation until the priority queue is empty: "remove the highest priority vertex from the queue, then add to the queue all vertices adjacent to that vertex (update the priority of all those which are already on the queue, if warranted)." Graph traversal methods differ in the way in which the priorities are updated and in the way in which the priority queue is implemented. (See [Sedgewick, 83] or [Tarjan, 83] for more details and examples.) An end product of the traversal is a spanning tree (if the graph is connected). Actually, it is convenient to think of the vertices as being divided into three classes during the execution of the algorithm: *spanning tree* vertices (which have been removed from the priority queue and added to the spanning tree); *unseen* vertices (which haven't been encountered at all); and *fringe* vertices (those on the priority queue: these are adjacent to spanning tree vertices but haven't yet been visited).

The classical *shortest path* [Dijkstra, 59] algorithm results from assigning to each fringe vertex a priority equal to its distance in the spanning tree to the start vertex. In this case, the spanning tree is a "shortest path tree" which defines the shortest path in the graph from the start vertex to each other vertex. To find the shortest path from s to t, we assign a high priority to s at the beginning to make it the start vertex, then stop the algorithm when t (the *destination* vertex) is encountered. To simplify discussion, we'll assume that the destination vertex is the farthest from the start, so that the classical algorithm must examine every node and edge in the graph. Figure 1 pictures the spanning tree, fringe, and

* Support for the first author was provided in part by NSF Grant MCS-83-08806. Support for the second author was provided in part by NSF Grant MCS-81-05324, by an IBM research contract, and by an IBM Faculty Development Award. Support for this research was also provided in part by ONR and DARPA under Contract N00014-83-K-0146 and ARPA Order No. 4786. Equipment support was provided by NSF Grant MCS-81-218106.

unseen vertices after the classical (Dijkstra) algorithm is run on a typical graph.

The priority queue can be implemented as a Fibonacci heap ([Fredman and Tarjan, 84]) for a total running time of $O(V \log V + E)$. (Here E and V are the number of edges and vertices, respectively, in the graph. Generally we assume that the graph is likely to be connected; that is $E > V$. Roughly, we say that a graph with $E = \Omega(V^2/\log V)$ is *dense*; a graph with $E = O(V \log V)$ is *sparse*.) Of course, the time required for input of a graph with E edges is $O(E)$, so the Dijkstra algorithm is optimal or near-optimal when input time is included. However, if shortest paths are to be found for several pairs of vertices in the same graph, or if the graph is already represented in memory for some other reason (which is typical in applications), then algorithms which run in $o(E)$ steps are of interest.

The basic idea ([Hart, Nilsson, and Raphael 68]) for improving the performance of the Dijkstra algorithm for Euclidean graphs is simple: to compute the shortest path tree, use the above method but assign each vertex priority according to the distance in the tree to the start vertex (as before) *plus* the Euclidean distance to the destination vertex. Thus, we use global information about the graph to guide the search. The Euclidean distance from each vertex to the destination is a lower bound on the distance in the graph to the destination, so we still are ensured that we have found the shortest path to the destination when it is first reached. Specifically, we define $Eucl\_dist(x, y)$ to be the Euclidean straight-line distance between x and y, and we define $dist(x, y)$ to be the length of the shortest path in the graph from vertex x to vertex y. If we assign priorities so that we always choose the fringe vertex x that minimizes the sum

$$dist(s, x) + Eucl\_dist(x, t),$$

we get an algorithm that runs much faster than the standard graph algorithms on typical graphs. We call this algorithm the *Euclidean heuristic*. The algorithm is fast because the search can be terminated when t is added to the spanning tree; the correctness is based on the fact that the $Eucl\_dist(x, t)$ term gives a lower bound on $dist(x, t)$. As with the standard graph algorithms, the shortest path from s to t can be output in reverse order, by storing back pointers at each spanning tree node.

The classical shortest path algorithm examines every node in the graph; the proposed heuristic could do the same, so there is no improvement in worst-case performance. On the other hand, it would seem that for many graphs that might be encountered in practice, the actual shortest path might be discovered long before every vertex is encountered. A typical case is illustrated in Figure 1. In this paper, we consider the question of determining exactly how much might be saved with the Euclidean heuristic.

To properly model Euclidean graphs for purposes of analysis it is necessary to consider not only the connections among vertices but also point set placements (which specify exactly where the vertices are). To fix ideas, we assume that the vertices are randomly distributed points in the unit square and that s and t are situated at the corners: $s = (0, 0)$, $t = (1, 1)$. Then we consider various models for assigning edges to the vertices. Surprisingly, for many natural graph models, the actual point placement is not particularly relevant to the analysis. Section 2 gives details on such models and a general result which establishes an $O(V)$ bound on the running time. For other models, which are perhaps more realistic, the point placement plays a crucial role. We have had some success establishing specific results for such models and have identified several techniques which seem germane to the analysis. More details on this are given in Section 3.

2. ANALYSIS FOR RANDOM GRAPHS

In this section we'll investigate the performance of the Euclidean heuristic for six models of random graphs on V vertices. Each model has a parameter that determines the expected density of the edges in the graph. The surprising result is that the average running time for each model is $O(V)$ and is independent of the expected density and of the placement of the vertices. This represents a significant savings over the classical algorithms for general graphs.

Definition 1. The six models of random graphs on V vertices that we consider in this section are defined as follows. The vertices can be placed in the unit square in any way, since their locations do not affect the running time of the Euclidean heuristic.

1. Each of the $V(V-1)/2$ possible edges appear in the graph with probability $p(V)$, independent of all the other edges.
2. The graph has $E(V)$ edges, with each of the $\binom{V(V-1)/2}{E}$ sets of E edges equally likely.
3. The graph is directed. Each of the $V(V-1)$ possible directed edges appear in the graph with probability $p(V)$, independent of all the other edges.
4. The graph is directed and has $E(V)$ edges, with each of the $\binom{V(V-1)}{E}$ sets of E edges equally likely.
5. The graph is directed. Each vertex has in-degree $k(V)$, with each of the $\binom{V-1}{k}$ sets of k in-edges equally likely.
6. The graph is directed. Each vertex has out-degree $k(V)$, with each of the $\binom{V-1}{k}$ sets of k out-edges equally likely.

The *expected density* for the graph model is the expected percentage of edges present: the expected number of edges divided by $V(V-1)/2$ for the undirected case, by $V(V-1)$ for the directed case.

The expected densities for the six models are, respectively, p, $2E/(V(V-1))$, p, $E/(V(V-1))$, $k/(V-1)$, and $k/(V-1)$. The expected number of edges for the six models is $pV(V-1)/2$, E, $pV(V-1)$, E, kV, and kV. Recall once again that the classical algorithms examine each edge in the graph, so that these expressions give lower bounds on the running time for the classical methods.

Theorem 1. *The average running time of the Euclidean heuristic for each of the six models of random graphs given above is $O(V)$, independent of the expected density and of the placement of the vertices.*

The average running time of the fastest known algorithm for general non-Euclidean graphs ranges from $O(V^2)$ for dense graphs (for which the expected density is a constant) to $O(V \log V)$ for sparse graphs (for which the expected density is $\Theta((\log V)/V)$). The Euclidean heuristic results in significant savings.

Proof. First we consider model *3*. The Euclidean heuristic will find the optimum path during the next iteration if and only if the fringe vertex that is added to the spanning tree during the current iteration is connected to t via a directed edge. The probability of this happening is p. Hence, we can model the number of iterations I of the heuristic by a geometrically distributed random variable with mean $1/p$. This model actually gives an upper bound on I, because the algorithm can halt prematurely if s and t are not in the same connected component. The exact average value of I is $1/p$ minus some second-order terms.

A lower bound on the expected running time of the Euclidean heuristic is the average number of edges examined, which can be shown to be $O(IpV) = O(V)$ by an application of Wald's Lemma (see [Loève, 77]). A priority queue is used to store the vertices on the fringe. Normally the overhead of maintaining the priority queue would introduce an extra $\log V$ term into the running time, which would make the running time $O(V \log V)$. However, we can reduce the average running time to $O(V)$ by implementing the priority queue in one of two ways.

The first way is to use a heap *with duplicate entries*. Whenever a vertex is moved from the fringe to the spanning tree, an entry is added to the heap for the cost of each of the vertex's neighbors that is not already in the tree and that does not have a higher-priority entry already in the heap. The insertions into the heap are done in a bottom-up manner in $O((\#\text{ entries}) + \log V)$ steps. By Walds's Lemma, the total cost of such insertions over the course of the algorithm is I times this quantity, which is $O(IpV + I \log V) = O(V)$, on the average. There may be multiple entries in the heap for the same vertex; the heap contains (possibly duplicate) entries for the fringe vertices as well as some duplicate entries for vertices that have already been put in the spanning tree. Thus, when the highest priority fringe vertex must be selected, several duplicate entries might first have to be removed from the top of the heap. Fortunately, we can show that, on the average, each spanning tree vertex has only a constant number of duplicate entries, which means that selecting the highest priority fringe vertex requires $O(1)$ deletions from the top of the heap and can be done in $O(\log V)$ expected time. The average total cost of selections is thus $O(I \log V) = O(V)$. This gives the desired $O(V)$ time bound. More details will appear in the full paper.

An alternative to the heap with duplicate entries is the Fibonacci heap introduced in [Fredman and Tarjan, 84]. Selecting the highest priority vertex costs $O(\log V)$ time, but insertions of new vertices and updates in which the priority is increased can be done in only constant time. The $O(V)$ time bound for the Euclidean heuristic follows by the same reasoning as above. The Fibonacci heap priority queue is general-purpose and more sophisticated than heaps with duplicate edges; however, the heap implementation is much simpler to code for this application.

The other models can be handled similarly. The analysis of model *5* involves the distribution of the *skip random variable* S discussed in [Vitter, 83]: the number of iterations I can be modeled by the minimum of k random integers picked without replacement from the $V-1$ integers $\{0, 1, 2, \ldots, V-2\}$. The average value is $(V-k-1)/(k+1)$, which is approximately the inverse of the expected density. The rest of the analysis proceeds as before. ∎

The same result holds for dimensions higher than 2. It is certainly somewhat counterintuitive that the average case performance for the Euclidean heuristic should be independent of both the density and of the node placement. This is partly due to characteristics of the models: some alternative models are described in the next section. But also, this is indicative of the general applicability of the method: the essential point is that we have a reliable and easily computed lower bound (the Euclidean distance) for controlling the search. More comments on this may be found in Section 4.

3. MORE GRAPH MODELS AND TECHNIQUES

It can be argued that certain of the models above might be appropriate for studying certain specific applications areas, but not at all for others. For example, models *5* or *6* might be appropriate for studying airline route maps, but none of the models are entirely satisfactory for studying integrated circuits or railroad route maps, which are likely to lead to graphs which are near-planar and have few long edges. Some possibilities for modeling such situations are indicated below.

7. The V vertices are randomly distributed in the unit square. Each vertex is connected to all the vertices

within a radius of $r(V)$.

8. The V vertices are randomly distributed in the unit square. Each vertex is connected to its $k(V)$ nearest neighbors.
9. The V vertices are randomly distributed in the unit square. The edges are taken from the Delauney triangulation or one of its generalizations.
10. The V vertices are randomly distributed in the unit square. The edges form a "random" triangulation (if that can be made precise).

In this section, we introduce several useful approaches to the analysis of models like the ones above. The first approach we discuss shows that the average running time for model 7 is $O(V)$ and is independent of the radius r. This model is frequently used in the average-case analysis of heuristics for the travelling salesman problem.

For model 8 with $k = O(1)$ and for models 9–10, the number of edges in the graph is $O(V)$, so the classical algorithms would seem to perform quite well. However, it seems that the running time for the Euclidean heuristic on such graphs is *sublinear*. Research on this question is in progress.

Before we begin discussion of analysis techniques, let us mention one lemma that is central to all the techniques.

Lemma 1. *At the end of the Euclidean heuristic algorithm, the spanning tree consists of all those points x such that*

$$dist(s,x) + Eucl\_dist(x,t) < dist(s,t).$$

The points x for which equality holds are also in the spanning tree, in the worst case.

Proof. Straightforward from an examination of the algorithm. ∎

The priority queue implementations given in the proof of Theorem 1 can be used to yield the time bound

$$O\big((\text{spanning tree size})\log V + (\#\text{ edges processed})\big)$$

for the Euclidean heuristic. In all the models we shall analyze in this section, we have

$$(\text{ave. }\#\text{ edges processed}) = (\text{spanning tree size}) \times (\text{expected density}) \times V$$

$$(\text{expected density}) = \Omega\left(\frac{\log V}{V}\right).$$

Substituting these two expressions into the above bound, we can bound the average running time of the Euclidean heuristic by

$$O\big((\text{ave. spanning tree size}) \times (\text{expected density}) \times V\big). \qquad (\star)$$

The techniques given below bound the expected running time by obtaining a bound on the average spanning tree size.

Channels. The first approach we shall study considers paths that lie entirely withing narrow confines we call *channels*. We show that with high probability such a path exists. This implies a certain bound on the expected spanning tree size.

In the first example, we apply this technique toward the analysis of random graph model 1 in the last section. For the simple case, the channels are in the shape of ellipses. Consider the ellipse $A(d)$ defined by the set of all points r such that

$$Eucl\_dist(s,r) + Eucl\_dist(r,t) \le d.$$

We denote the number of vertices in the ellipse A by $\|A\|$. We set d to the minimum value $d \ge \sqrt{2}$ so that $A(d)$ contains a vertex x that is connected directly to both s and t. On the average, we have $\|A\| = 2 + 1/p(V)^2$. By Lemma 1, we know that the only points that can appear in the spanning tree are the vertices in A. By $(\star)$, the average running time is $O(\|A\|p(V)V) = O(V/p(V))$. For the dense case $p(V) = \Theta(1)$, the average running time is $O(\sqrt{E})$.

The bound on the average number of edges which follows from the channel argument, $O(V/p(V))$, is asymptotically less than $E = O\big(V^2p(V)\big)$ when $p(V) = \omega(V^{-1/2})$. We can strengthen the channel argument to show that the Euclidean heuristic runs in $o(E)$ average time for much smaller values of $p(V)$, namely, when $p(V) = \omega(V^{-(k-1)/k})$, for any fixed k. The approach is to redefine the channel A to be the symmetric region of largest area such that for *any* set of $k-1$ points in A, the shortest path from s to t that goes through the $k-1$ points has length $\le d$. We call A the *inner channel*. We also define the *outer channel* B to be the ellipse $B(d)$. We set d to the minimum value $d \ge \sqrt{2}$ such that there is a path from s to t containing k edges that lies entirely within A. We can show that $\|B\| = O(k\|A\|) = O(\|A\|)$, since k is a constant, and $\|A\| = O(p^{-k/(k-1)})$, on the average. By $(\star)$, the average running time can be bounded by $O(\|B\|pV) = O(Vp^{-1/(k-1)})$. This is $o(E)$ when $p(V) = \omega(V^{-(k-1)/k})$.

The inner channel A in the above argument has many interesting properties and is deserving of study in its own right. For lack of a better name, we call such a region a *k-lipse*, since it generalizes the notion of an ellipse, which is the special case $k = 2$.

We can use channels in a different way to analyze model 7.

Theorem 2. *The average running time of the Euclidean heuristic for model 7 is $O(V)$.*

Proof. In this proof we make use of two types of channels. The first channel consists of disjoint rectangular boxes aligned perpendicular to the diagonal from s to t. The second channel is the ellipse (for the case $k = 2$) discussed above.

We assume that $E = \Omega(V \log V)$, which implies that $r = \Omega(\sqrt{(\log V)/V})$. Let us divide the diagonal from s to t into segments of length ℓ, for some real constant $\ell \geq 0$ such that $\sqrt{2}/\ell$ is an odd integer and $\ell < r/4$. We shall use a "diagonal" coordinate system, in which each point x is represented by a distance $d_1(x)$ from s along the diagonal from s to t and by a distance $d_2(x)$ perpendicular to the diagonal. Let $n = \frac{1}{2}(\sqrt{2}/\ell + 1)$. We define the *box* $B_{i,j}(h)$, for $1 \leq i \leq n$, $-n < j < n$, and $0 \leq h \leq (\ln V)/(\ell V)$, to be the set of points x in the unit square such that $(2i-2)\ell \leq d_1(x) \leq (2i-1)\ell$ and $(j - \frac{1}{2})h \leq d_2(x) \leq (j + \frac{1}{2})h$. Each $B_{i,j}(h)$ is a box parallel to the diagonal with length ℓ and height h.

From the above definitions, it is easy to show that when V is large enough every point in $B_{i,j}(h)$ is within a radius of r of every point in $B_{i+1,j\pm1}(h)$; thus, every pair of vertices in $B_{i,j}(h)$ and $B_{i+1,j\pm1}(h)$ is connected by an edge.

We shall say that there is a path in the channel from s to t if if there is a sequence of boxes $B_{1,j_1}(h)$, $B_{2,j_2}(h)$, $\ldots B_{n,j_n}(h)$ such that each box contains at least one vertex and such that $j_1 = 0$ and $j_{k+1} = j_k \pm 1$, for $1 \leq k < n$. The path in the channel from s to t consists of $n-1$ edges with total length $\leq \sqrt{2} + 2h^2/\ell^2$. When there is a path in the channel, we can use Lemma 1 to bound the spanning tree size by the number vertices in the ellipse $B(\sqrt{2} + 2h^2/\ell^2)$, which on the average is $O(hV/\ell)$.

Let us use $P(h)$ to denote the probability that there is a path in the channel from s to t. Taking conditional expectations, we can bound the average spanning tree size by

$$O\left(\int_0^{(\ln V)/(\ell V)} \frac{hV}{\ell} P'(h)\,dh\right) + O\left(V\left(1 - P\left(\frac{\ln V}{\ell V}\right)\right)\right).$$

By the product rule, this becomes

$$O\left(\frac{V}{\ell}\left(hP(h)\Big|_0^{(\ln V)/(\ell V)} - \int_0^{(\ln V)/(2\ell V)} P(h)\,dh\right)\right) + O\left(V\left(1 - P\left(\frac{\ln V}{\ell V}\right)\right)\right). \quad (**)$$

We will now show that we can bound $P(h)$ by

$$P(h) \geq 1 - 2\sum_{0 \leq k \leq n/2} k(k+1)5^k q^{k+1}, \quad (***)$$

where $q < (1-h\ell)^V < e^{-h\ell V}$ is the probability that a given box $B_{i,j}(h)$ contains no vertices. We derive $(***)$ as follows: If there is no path from s to t in the channel, then there must be an "antipath" of empty boxes that separate s from t. Let k, for $0 \leq k \leq n/2$, be the number of nonempty boxes in the longest contiguous chain of nonempty boxes in the channel starting at s. There are k choices for the last nonempty box in the chain. The antipath must contain at least $k+1$ empty boxes. At least one of the nonempty boxes must be adjacent to the last nonempty box in the chain. The remaining k boxes in the antipath can be chosen iteratively in $\leq k5^k$ ways. The probability of a given set of $k+1$ boxes being empty is bounded by q^{k+1}. (The bound is strict when $k > 1$.) Hence, the probability that there is no path in the channel from s to t and that the longest chain of nonempty boxes in the channel contains k nonempty boxes, for $0 \leq k \leq n/2$, is bounded by $k(k+1)5^k q^{k+1}$. Summing up on k and by symmetry for the k in the range $n/2 < k \leq n-1$, we find that the probability of no path in the channel from s to t is $\leq 2\sum_{0 \leq k \leq n/2} k(k+1)5^k q^{k+1}$. This proves $(***)$.

By $(***)$ we have

$$P\left(\frac{\ln V}{\ell V}\right) = 1 - O\left(\frac{1}{V}\right)$$

and

$$\int_0^{(\ln V)/(\ell V)} P(h)\,dh \geq \int_{2/(\ell V)}^{(\ln V)/(\ell V)} P(h)\,dh$$
$$\geq \left(\frac{\ln V}{\ell V} - \frac{2}{\ell V}\right) - \int_{2/(\ell V)}^{(\ln V)/(\ell V)} \sum_{0 \leq k \leq n/2} 2k(k+1)5^k q^{k+1}\,dh$$

We can bound the integral by

$$\sum_{0 \leq k \leq n/2} 2k(k+1)5^k \int_{2/(\ell V)}^{(\ln V)/(\ell V)} q^{k+1}\,dh$$
$$= \sum_{0 \leq k \leq n/2} 2k(k+1)5^k \int_{2/(\ell V)}^{(\ln V)/(\ell V)} e^{-(k+1)h\ell V}\,dh$$
$$\leq \sum_{0 \leq k \leq n/2} 2k(k+1)5^k \frac{e^{-2(k+1)}}{\ell V(k+1)}$$
$$= O\left(\frac{1}{\ell V}\right)$$

Substituting our bounds into $(**)$, we can bound the average spanning tree size by

$$O\left(\frac{V}{\ell}\left(\frac{\ln V}{\ell V} - \frac{\ln V}{\ell V} + \frac{1}{\ell V}\right) + V\left(\frac{1}{V}\right)\right) = O\left(\frac{1}{r^2}\right).$$

The expected density is $\Theta(r^2)$. By $(\star)$, the average running time for the Euclidean heuristic is bounded by

$$O\left(\frac{1}{r^2} r^2 V\right) = O(V). \quad \blacksquare$$

Sizing up the spanning tree. Another approach for analyzing the running time of the Euclidean heuristic is

to use properties of the graph model to characterize the number of points in the spanning tree.

Let's denote the average length of the shortest path by ℓ. Our intuition is that the average spanning tree size at the end of the algorithm will be the expected number of points x for which the inequality in Lemma 1 holds, with the right-hand side replaced by ℓ.

We can get estimates on the number of points in the spanning tree by first determining the relative length of $dist(x,y)$ vs. $Eucl\_dist(x,y)$. Suppose we can show that for all vertices x, the average distance from s to x is roughly

$$dist(s,x) \approx r(V) \times Eucl\_dist(s,x),$$

where $r(V) \geq 1$. Then by Lemma 1, the spanning tree consists of those vertices x such that

$$r(V) \times Eucl\_dist(s,x) + Eucl\_dist(x,t) \leq r(V) \times Eucl\_dist(s,t).$$

To make this more rigorous, we need to specify the rate at which the approximation becomes "good," by showing convergence in probability (see [Loève, 77]). We will bound the ratio of *dist* to *Eucl_dist* by $r_1(n)$ from above and $r_2(n)$ from below.

Theorem 3. *Suppose that the graph is a random instance of a graph from some model in which the locations of the V vertices are uniformly distributed. We define an "upper" bound $r_1(V)$ and a "lower" bound $r_2(V)$ with $1 \leq r_2(V) \leq r_1(V) \leq c$, for some constant c. We also have a function $h(V)$ in the range $0 \leq h(V) \leq 1$ such that $\lim_{V\to\infty} h(V) = 0$. Suppose that*

$$\Pr\{dist(s,x) \leq r_1(V) \times Eucl\_dist(s,x)\} > 1 - h(V),$$
$$\Pr\{dist(s,x) \geq r_2(V) \times Eucl\_dist(s,x)\} > 1 - h(V),$$

where the probabilities are averaged over all vertices $x \neq t$ and over all instances of the graph. Suppose the same inequalities also hold for the case $x = t$. Then the expected size of the spanning tree at the end of the Euclidean heuristic can be bounded by

$$O\left((1-h(V))V\sqrt{r_1(V)-1} + h(V)V\right)$$

and

$$\Omega\left((1-h(V))V\frac{(r_2(V)-1)^2}{(r_1(V)-1)^{3/2}}\right).$$

Proof. Let us prove the upper bound first. Consider a random instance of a graph. With probability $1 - h(V)$, we have

$$dist(s,x) \leq r_1(V) \times Eucl\_dist(s,x),$$
$$dist(s,x) \geq r_2(V) \times Eucl\_dist(s,x),$$

for all $x = t$ and for some fraction $1 - O(h(V))$ of the values $x \neq t$. Combining this with Lemma 1, we can bound the expected spanning tree size by

$$(1-h(V))V\,|A| + h(V)V,$$

where $|A|$ is the area of the region A that consists of the points x in the unit square satisfying

$$r_2(V) \times Eucl\_dist(s,x) + Eucl\_dist(x,t) \leq r_1(V) \times Eucl\_dist(s,t).$$

We can enclose A by the region A' defined by the inequality

$$Eucl\_dist(s,x) + Eucl\_dist(x,t) \leq r_1(V) \times Eucl\_dist(s,t).$$

Let us convert to a "diagonal" coordinate system, in which each point x in the unit square is described by its distance $d_1(x)$ from s along the diagonal and by its height $d_2(x)$ perpendicular to the diagonal. By some algebraic manipulation, we can show that each $x \in A'$ satisfies $d_2(x) \leq \sqrt{(r_1(V)-1)/2}$. Hence, we have $|A'| \leq 2\sqrt{r_1(V)-1}$, which proves the upper bound.

The lower bound is more interesting. By the same reasoning as above, we can bound from below the average number of vertices in the spanning tree by

$$(1-h(V))V\,|B| + h(V)\cdot 0,$$

where B is the region consisting of the points x in the unit square satisfying

$$r_1(V) \times Eucl\_dist(s,x) + Eucl\_dist(x,t) \leq r_2(V) \times Eucl\_dist(s,t).$$

By algebraic manipulation, we can show that each $x \in B$ must satisfy $d_1(x) = O\big((r_2(V)-1)/(r_1(V)-1)\big)$. For a constant fraction of points in this range, $d_2(x)$ can be $\Omega\big((r_2(V)-1)/\sqrt{r_1(V)-1}\big)$. Hence, we have $|B| = \Omega\big((r_2(V)-1)^2/(r_1(V)-1)^{3/2}\big)$. The lower bound follows. ∎

Theorem 3 provides a fairly detailed statement about the average spanning tree size of the Euclidean heuristic in terms of basic properties of the graph model. In particular, the average spanning tree size is $o(E)$ if and only if there exists a function $r_1(V)$ such that $\lim_{V\to\infty} r_1(V) = 1$. This seems to be a promising approach (e.g. the function $r_1(V) \to 1$ exists for models *1–7*, but it is still open whether $r_1(V) \to 1$ for graph models *8–10*.

Worst-Case Graph Models. The following grid graph model is a worst-case graph for the Euclidean heuristic, because all paths from s to t have the same length, namely, 2.

11. The V vertices are arranged in a $\sqrt{V}$ by $\sqrt{V}$ array. Each vertex is connected to its immediate neighbors. (We refer to this as a "grid graph.")

By Lemma 1, the spanning tree will consist of all V vertices. If we add the $V - 2\sqrt{V} + 1$ SW-NE diagonals into the graph, then the Euclidean heuristic will proceed directly along the shortest path (the main diagonal) from s to t. The spanning tree size will be $O(\sqrt{V})$. But if we remove a constant fraction of the diagonals from the graph, at random locations, we can show that the spanning tree size will again be worst-case $\Theta(V)$.

4. CONCLUSIONS AND OPEN PROBLEMS

A simple heuristic has been presented which allows shortest paths to be found in Euclidean graphs in significantly fewer steps than required by classical algorithms. For a large class of random graph models, the time required is $O(V)$, rather than $O\big((E+V)\log V\big)$, as is the case for the classical algorithms. Many other types of graph models might be considered, specifically those in which the placement of the points plays a more important role. We have presented techniques which might be useful in the analysis of such models.

We also expect to study this problem in more general terms. For example, the term $Eucl\_dist(x,t)$ is just one means of giving a lower bound on $dist(x,t)$. Artificial intelligence researchers have studied the problem of designing heuristics for finding shortest paths, under the assumption that lower bounds for $dist(x,t)$ can be computed efficiently. (See [Hart, Nilsson, and Raphael 68], for example.) The work of [Lawler, Luby, and Parker 83] has focused on reducing combinatorial problems to search problems, to which methods like the Euclidean heuristic can be applied. Perhaps our several approaches to the analysis of the Euclidean heuristic can shed light on these more general problems.

Another interesting problem deals with relaxing the condition that a lower bound to $dist(x,t)$ must be used. Instead we can look at a heuristic that uses an approximation to $dist(x,t)$. As before the heuristic terminates when the destination vertex t is added to the spanning tree. The resulting heuristic might not find the shortest path, but it usually finds one close to optimum. There is a tradeoff between the running time of the heuristic and how close the answer is to optimum. Using the approach taken in Theorem 3, suppose we have a graph model for which r_1 and r_2 approach a limit $r > 1$. An interesting open problem is to study the performance of the heuristic in which we use $r \times Eucl\_dist(x,t)$ as an approximation for $dist(x,t)$. For example, using a multiple $r = \sqrt{2}$ makes the running time for the grid graph $O(\sqrt{V}\log V)$ rather than $O(V\log V)$. Other heuristics, such as minimizing the number of "turns" in the path from s to t, where s and t are points on a street map, have been studied in [Elliott and Lesk, 82].

The results of this paper suggest that it might be useful to study other problems dealing with Euclidean graphs. For example, preliminary results indicate that the heap data structure with duplicate edges that we used to prove Theorem 1 can be used to find the minimum spanning tree in Euclidean graphs in $O(E)$ time, on the average, assuming a random graph model in which the vertices are uniformly distributed in the unit square. The best minimum spanning tree algorithms for general graphs run in $O(E\log^* V)$ time using the sophisticated Fibonacci heap data structure ([Fredman and Tarjan, 84]). Also, approximation algorithms or algorithms with good average-case performance for the traveling salesman problem and other intractable problems would seem to be somewhat easier to develop for Euclidean graphs than for general graphs, Euclidean point sets, planar graphs, or other types of graphs.

Acknowledgements. We would like to thank Bob Tarjan for providing helpful references to past work.

REFERENCES

1. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs, *Numerishe Mathematik, 1* (1959).
2. R. J. Elliott and M. E. Lesk. Route Finding in Street Maps by Computers and People, *Proc. AAAI-82 Natl. Conference in Artificial Intelligence*, Pittsburgh, PA (August 1982), 258–261.
3. M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Proc. 25th Annual Symposium on Foundations of Computer Science*, West Palm Beach, FL (October 1984).
4. P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*, 2 (July 1968), 100–107.
5. E. L. Lawler, M. G. Luby, and B. Parker. Finding Shortest Paths in Very Large Networks. *Proc. WG '83 Intl. Workshop on Graphtheoretic Concepts in Computer Science*, Osnabrück, West Germany (June 1983), 184–199.
6. M. Loève. *Probability Theory.* Volume I. Graduate Texts in Mathematics, Springer-Verlag, New York (fourth edition 1977).
7. R. Sedgewick. *Algorithms.* Addison-Wesley, Reading, MA (1983).
8. R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA (1983).
9. J. S. Vitter. Faster Methods for Random Sampling. *Communications of the ACM, 27*, 7 (July 1984), 703–718.

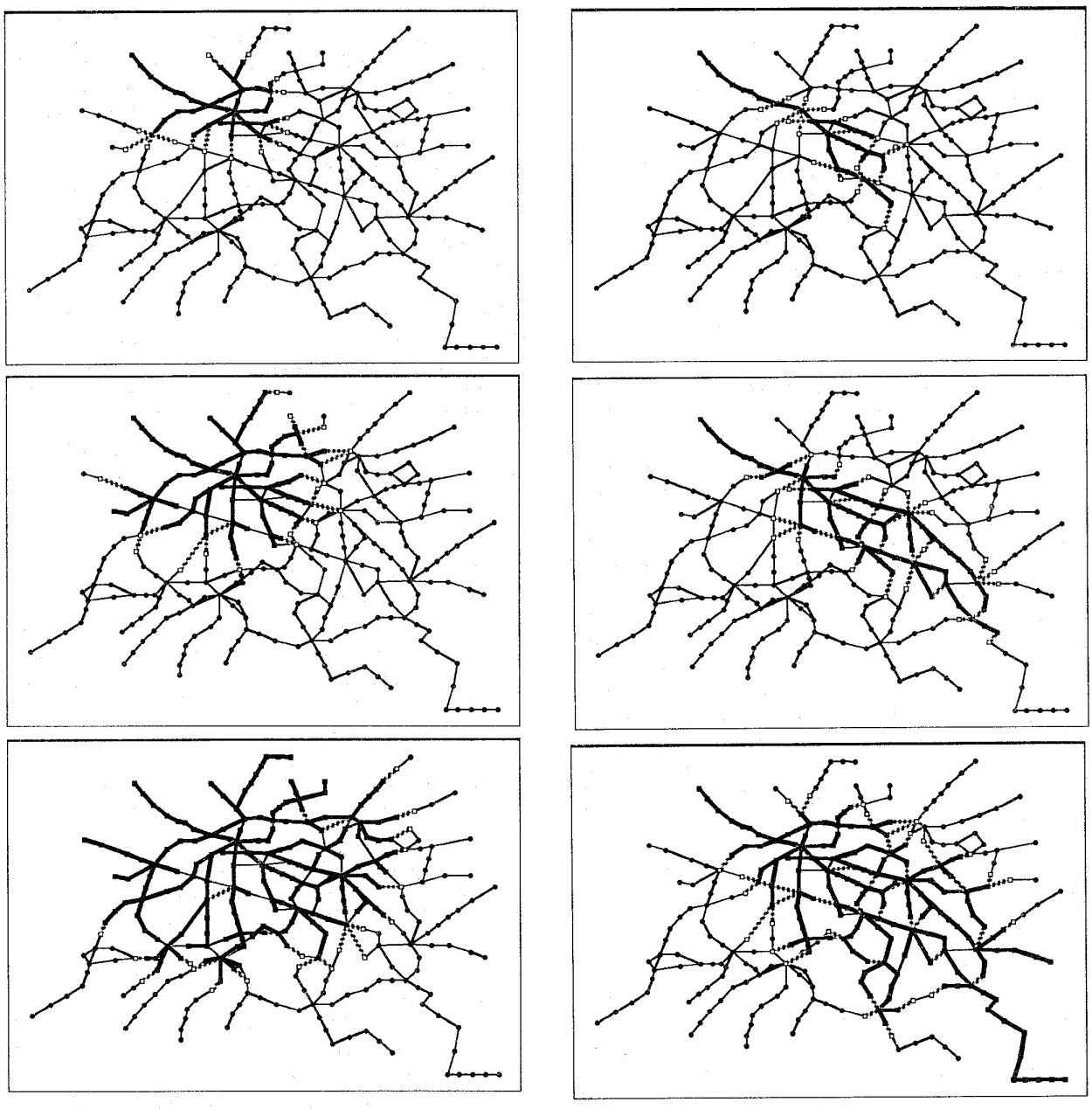

Figure 1. The pictures on the left show the decomposition of the vertices of a typical graph into spanning tree, fringe, and unseen vertices at various stages of Dijkstra's algorithm. The pictures on the right show the decomposition during the Euclidean heuristic. The start vertex s is in the top left-hand corner, and the destination vertex t is in the bottom right-hand corner. The solid boxes represent the spanning tree vertices; the heavy edges are the edges in the spanning tree. The unfilled boxes are the vertices in the fringe and are connected to the spanning tree via dashed lines. The small dots show the unseen vertices. For this graph, the Euclidean heuristic examines far fewer vertices and edges than does Dijkstra's algorithm.