Algorithmica (1986) 1:31–48

Algorithmica

Shortest Paths in Euclidean Graphs

Robert Sedgewick[1,2] and Jeffrey Scott Vitter[1,2]

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 Euclidean distances between points). For many graph models, the average 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 $O(E + V \log V)$ required by the classical algorithm due to Dijkstra.

Key Words. Analysis of Algorithms, Graph algorithm, Shortest path, Euclidean, Heuristic, Dijkstra's algorithm, Priority queue

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 characterize the precise improvement for a variety of random graph models through average-case analysis.

Specifically, we are interested in *Euclidean graphs*, where each vertex corresponds to a point in the d-dimensional Euclidean space $\mathfrak{R}^d$ and where the weight of an edge is proportional to the Euclidean straightline distance between the points it connects. Such graphs can 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. 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, one technique for finding the minimum spanning tree of a set of vertices in Euclidean space is to build a sparse subgraph S of the complete graph induced by the vertices, with the property that S is

[1] Department of Computer Science, Brown University, Providence, RI 02912, USA

[2] 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 Grants MCS-81-05324 and DCR-84-03613, an NSF Presidential Young Investigator Award, an IBM research contract, and an IBM Faculty Development Award. Support for this research was also provided in part by an ONR and DARPA under Contract N00014-83-K-0146 and ARPA Order No. 4786. Equipment support was provided by NSF Grant MCS-81-218106.

Received: April 4, 1985; revised August 5, 1985, Communicated by C. K. Wong.

guaranteed to contain the minimum spanning tree; then an algorithm for general graphs is used to compute the minimum spanning tree of S ([Yao, 82]). The underlying theme of this paper is that it is possible to gain efficiency by making use of the geometric information *directly within the graph traversal algorithm*.

Classical graph traversal algorithms can be developed according to the following general schema: Initially, one distinguished *start* vertex is placed into a priority queue, and a spanning tree is made empty. Then the **while**-loop given below is executed; each iteration removes one vertex from the priority queue and adds it to the spanning tree.

```
while not done do
    begin
    Remove the highest priority vertex $v$ from the priority queue;
    for each neighbor $w$ of $v$ in the graph do
        if $w \in$ priority queue then
            Reevaluate the priority of $w$
        else if $w \notin$ spanning tree then
            Insert $w$ into the priority queue;
    Add $v$ to the spanning tree, as a child of the vertex that set $v$'s priority
    end;
```

The loop halts when the priority queue is empty or when the appropriate stopping condition arises. 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 of some portion of the graph.

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 have not been encountered at all); and *fringe* vertices (those in the priority queue: these are adjacent to spanning tree vertices but have not yet been visited). The problem we consider in this paper is to find the shortest path between two designated vertices, the *start vertex* s and the *destination vertex* t. The length of a path is the sum of the weights of all the edges along the path.

The classical shortest path algorithm due to Dijkstra ([Dijkstra, 59]) results from assigning to each fringe vertex x a value equal to the shortest distance from the start vertex s to x, among all paths consisting of spanning tree edges followed by a single edge in the graph from the tree to x. The highest priority fringe vertex is the one with the lowest value. To find the shortest path from s to t, we assign a low value (high priority) to s at the beginning in order to make it the start vertex, then stop the algorithm when the destination vertex t is encountered. The resulting spanning tree is a "shortest path tree," which defines the shortest path in the graph from s to every vertex that is closer to s than t is. The left side of Figure 1 pictures the spanning tree, fringe, and unseen vertices at different stages during the execution of Dijkstra's algorithm on a typical graph.

The priority queue can be implemented as a Fibonacci heap ([Fredman and Tarjan, 84]) for a total running time of $O(E + V \log V)$, where E and V are the numbers of edges and vertices, respectively, in the graph. Of course, the time required for input of a graph with E edges is $\Omega(E)$, so Dijkstra's 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 that run in $o(E)$ steps are of interest.

The basic idea ([Hart, Nilsson, and Raphael, 68]) for improving the performance of Dijkstra's algorithm when run on Euclidean graphs is simple: to build the spanning tree, we use the above method, but assign to each fringe vertex x a value equal to the distance through the tree from s to x (as before) *plus* the Euclidean distance from x to t. Thus, we use global information about the graph to guide the search. To formalize this, we define $\mathit{Eucl\_dist}(x, y)$ to be the Euclidean (l^2-norm) distance between vertices v_1 and v_2, and we define $\mathit{dist}(v_1, v_2)$ to be the length of the shortest path in the graph from v_1 to v_2. The length of a path is equal to the sum of the weights of the edges along the path; the weight of edge $\langle v_1, v_2 \rangle$ is defined to be $\mathit{Eucl\_dist}(v_1, v_2)$. Each fringe vertex x is assigned the following value:

$$\min_w \{ \mathit{dist}(s, w) + \mathit{Eucl\_dist}(w, x) \} + \mathit{Eucl\_dist}(x, t).$$

The minimum is taken over all w such that $w \in$ spanning tree and $\langle w, x \rangle \in$ graph. The resulting algorithm, which we call the *Euclidean heuristic*, runs much faster than the standard graph algorithms on typical graphs, for two main reasons: (1) the spanning tree tends to grow in the direction of t, and (2) the search for the shortest path can be terminated as soon as t is added to the spanning tree. The correctness of (2) follows easily from the fact that the $\mathit{Eucl\_dist}(x, t)$ term gives a lower bound on $\mathit{dist}(x, t)$. 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.

To simplify discussion, we shall assume that t is the farthest vertex from s, so that Dijkstra's algorithm must examine every vertex and edge in the graph. The Euclidean heuristic could do the same, so there is no improvement in worst-case performance (although obviously the heuristic can never examine more vertices and edges than does Dijkstra's algorithm). On the other hand, it seems that for many graphs encountered in practice the actual shortest path can 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 the point set placements (which specify exactly where the vertices are). We typically assume that the vertices are independently and uniformly distributed points in the unit d-dimensional cube, for some $d \geq 2$, and that s and t are situated at the corners

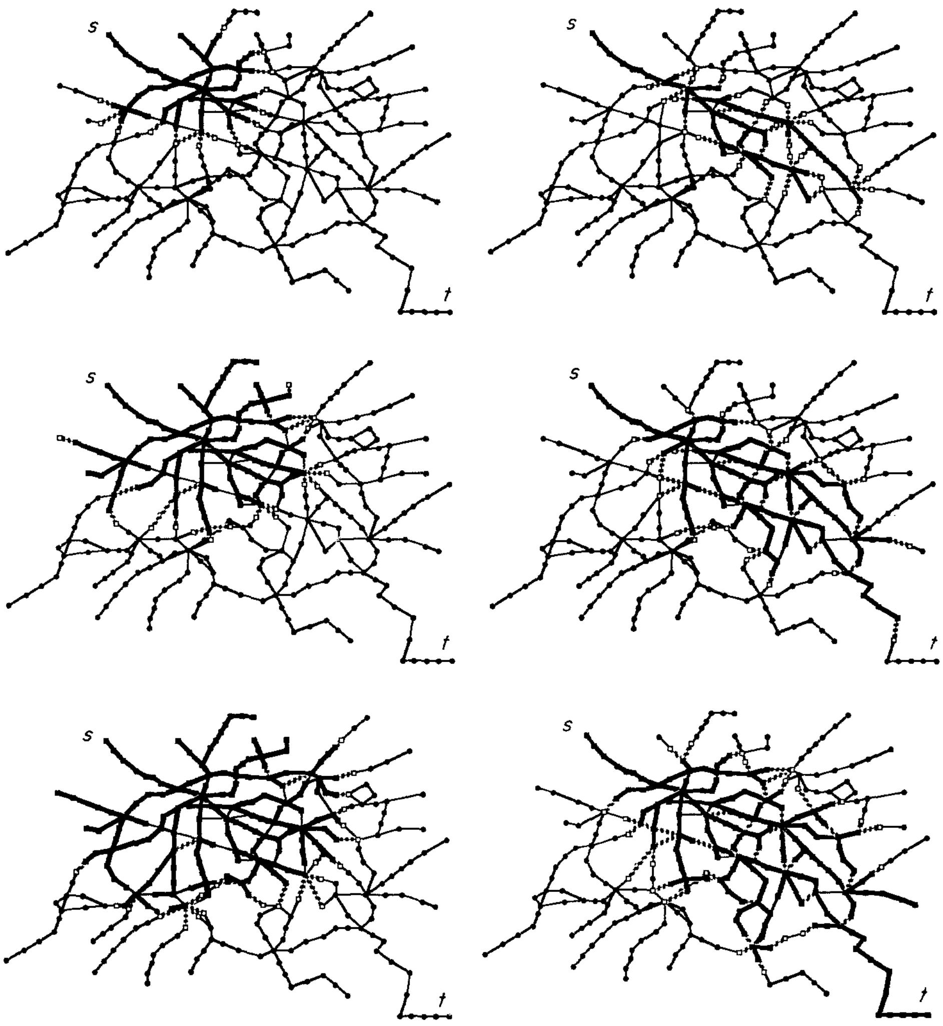

Fig. 1. The pictures on the left show the decomposition of the vertices of a typical graph into spanning tree, fringe, and unseen vertices at various stages of Dijkstra's algorithm. The pictures on the right show the decomposition during the Euclidean heuristic. The start vertex s is in the top left-hand 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. As the bottom pictures show, Dijkstra's algorithm is far from finished when the Euclidean heuristic terminates.

$s = (0, \dots, 0)$ and $t = (1, \dots, 1)$. We make t the farthest vertex in the graph from s in order to maximize the running time of the Euclidean heuristic. We consider various models for assigning edges to the vertices. Surprisingly, for many natural graph models, the actual point set 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 average 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. Conclusions, generalizations to metrics other than the Euclidean distance, and open problems are discussed in Section 4.

2. Analysis for Random Graphs.

In this section we shall 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 $O(V)$, independent of the expected density, the placement of the vertices, and the dimension d. 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 start and end vertices are $s = (0, \dots, 0)$ and $t = (1, \dots, 1)$. The remaining $V - 2$ vertices can be placed in the unit d-dimensional cube in any way, since their locations do not affect the running time of the Euclidean heuristic.

1. The graph is undirected. 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 is undirected. 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, which we assume is $\Omega(V \log V)$ in order to make the graph connected with high probability. 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.

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

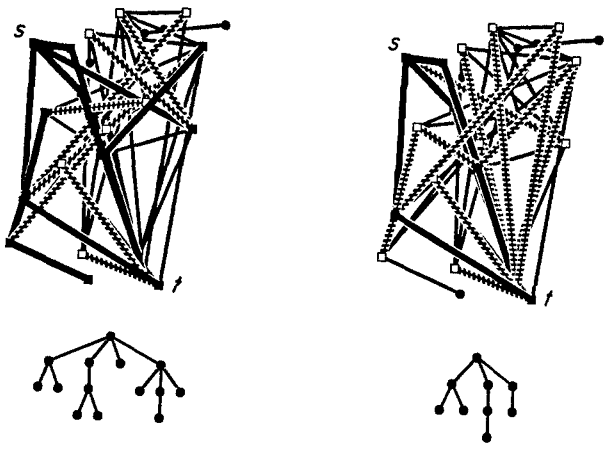

Fig. 2. The final result of Dijkstra's algorithm (on the left) and the Euclidean heuristic (on the right) on a random graph. The same remarks as in Figure 1 apply. The spanning tree constructed in each case is pictured below the graph. The Euclidean heuristic products a "skinnier" spanning tree than does Dijkstra's algorithm, reflecting the fact that its search for the shortest path is more focused. The Euclidean heuristic runs in $O(V)$ time, on the average, independent of the expected density of the graph, the placement of the vertices, and the dimension.

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, as illustrated in Figure 2.

THEOREM 1. The average running time of the Euclidean heuristic for each of the six models of random graphs given above is $O(V)$. The coefficient implicit in the $O(V)$ bound can be chosen independently of the expected density, the placement of the vertices, and the dimension d.

A lower bound on the expected running time of the Euclidean heuristic is the average number of edges examined. 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 $\Omega((\text{ave. \# edges examined})\log V)$.

However, we can reduce the average running time by implementing the priority queue as a Fibonacci heap ([Fredman and Tarjan, 84]). Selecting the highest priority vertex and deleting it from the priority queue 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. This gives us the time bound

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

for the Euclidean heuristic. For the graph models we consider in Theorems 1 and

2, we have

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

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

Formula (1) follows from Wald's Lemma (see [Loève, 77]). Substituting these two expressions into the above time bound, we find that the average running time of the Euclidean heuristic is O(ave. # edges processed), which can be computed via (1). In Theorems 1 and 2, we bound the expected running time by obtaining a bound on the average spanning tree size (or equivalently on the average number of iterations).

Proof of Theorem 1. 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 (and the final spanning tree size) by a geometrically distributed random variable with mean $1/p$. This model actually gives an upper bound on the average number of iterations $\bar{I}$, because the algorithm can halt prematurely if s and t are not in the same connected component. By (1), the average running time of the Euclidean heuristic is $O(\bar{I}pV) = O(V)$.

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. ■

An alternative to using a Fibonacci heap as the priority queue is to use a *mergeable 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. Inserting t items into the heap can be done in a batched bottom-up manner in $O(t + \log V)$ steps, as follows: First, an auxiliary 2–3 heap of the t items is formed in $O(t)$ time using the standard heapify algorithm. Then it is merged with the main 2–3 heap in $O(\log V)$ time, using the algorithm given in [Aho, Hopcroft, and Ullman, 74]. The average number of inserted items t is bounded by pV. By reasoning similar to (1), the total cost of such insertions over the course of the algorithm is $O(\bar{I}pV + \bar{I} \log V) = O(V)$, 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. The analysis in [Gonnet, 81] shows that the maximum number of duplicate entries of the same vertex, on the average, is asymptotically $\log(IpV)/\log\log(IpV) = O(\log V/\log\log V)$. By Wald's Lemma, we can bound the total number of duplicate entries that come to the top of the heap by $O(\bar{I}\log V/\log\log V)$. The average total cost of selections is thus $O(\bar{I}(\log V/\log\log V)\log V) = O(V)$, when $p = \Omega((\log V)^2/(V\log\log V))$. We conjecture that the $O(V)$ bound also holds for smaller p.

It is certainly somewhat counterintuitive that the average case performance for the Euclidean heuristic should be independent of the expected density, the node placement, and the dimension. This is partly due to characteristics of the models, but also it is indicative of the general applicability of the method. Some alternative models are described in the next section.

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. In each case, the start and end vertices are $s = (0,0)$ and $t = (1,1)$; the remaining $V - 2$ vertices are assumed to be independently and uniformly distributed in the unit square.

7. Each vertex is connected to all the vertices within a radius of $r(V)$.
8. Each vertex is connected to its $k(V)$ nearest neighbors.
9. The edges are taken from the Delauney triangulation or one of its generalizations.
10. 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 appears 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. ■

Channels. The first approach we shall study considers paths that lie entirely within 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.

For our first example, we apply this technique toward the analysis of random graph model 1 in the last section for dimension $d = 2$, with the additional assumption that the vertices are independently uniformly distributed in the unit square. For the simple case, the channels are in the shape of ellipses. Consider the ellipse $A(r)$ defined by the set of all points x such that

$$\mathit{Eucl\_dist}(s, x) + \mathit{Eucl\_dist}(x, t) \leq r.$$

We denote the number of vertices in the ellipse A by $\|A\|$. We set r to the minimum value $r \geq \sqrt{2}$ so that $A(r)$ contains a vertex x that is connected directly to both s and t. For a given vertex x, the probability that the edges $\langle s, x\rangle$ and $\langle x, t\rangle$ are both present is p^2. Hence, we have $\|A\| = 2 + 1/p^2$, on the average.

By Lemma 1, we know that the only points that can appear in the spanning tree are the vertices in A. By (1), the average running time is $O(\|A\| pV) = O(V/p)$. For the dense case $p(V) = \Theta(1)$, the average running time is $O(\sqrt{E})$.

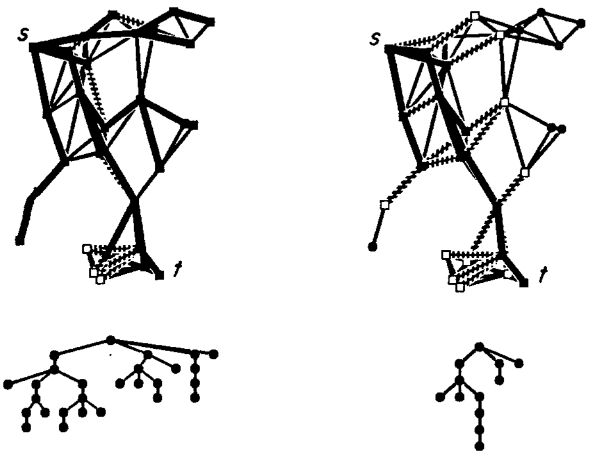

Fig. 3. The final result of Dijkstra's algorithm (on the left) and the Euclidean heuristic (on the right) for a typical graph in model 7, in which the vertices are randomly placed and two vertices are connected by an edge iff they are within a radius $r(V)$ from one another. The same remarks as in Figure 1 apply. The spanning trees constructed by both algorithms are pictured below; as in Figure 2, the spanning tree constructed by the Euclidean heuristic contains significantly fewer nodes. The Euclidean heuristic runs in $O(V)$ time, on the average, independent of the radius $r(V)$.

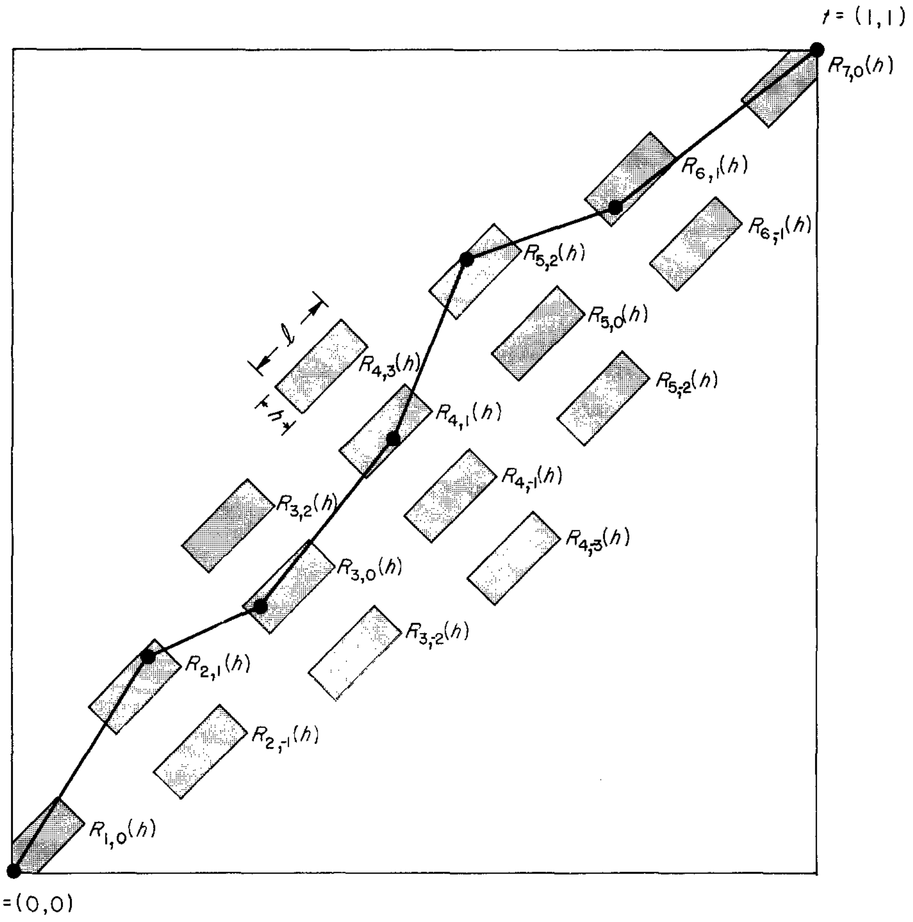

Fig. 4. A path in the channel, defined on the diagonal coordinate system used in the proof of Theorem 2, for the case $n = 7$. Each rectangle has length l and height h.

We can use channels in a more powerful way to analyze the performance of the Euclidean heuristic for graph model 7, in which the vertices are independently and uniformly distributed in the unit square and each pair of vertices within a distance r from one another are connected via an edge. Figure 3 compares the performances of the Euclidean heuristic and Dijkstra's algorithm for a typical graph of model 7.

THEOREM 2. The average running time of the Euclidean heuristic for model 7 is $O(V)$. The coefficient implicit in the $O(V)$ bound can be chosen independently of the expected density of the graph.

PROOF. In this proof we make use of two types of channels. The first channel consists of disjoint rectangles aligned parallel to the diagonal from s to t. The second channel is the ellipse discussed above.

The average number of edges $\bar{E}$ in the graph is $\Theta(r^2)$. As in Theorem 1, we assume that $\bar{E} = \Omega(V \log V)$, which implies that $r = \Omega(\sqrt{(\log V)/V})$. Let us divide the diagonal from s to t into segments of length $l > 0$, where l is as large as possible a real number such that $\sqrt{2}/l \equiv 1 \pmod 4$ and $l < r/4$. In particular, we have $l = \Theta(r)$. 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 $n = \frac{1}{2}(\sqrt{2}/l + 1)$. We define the *rectangle* $R_{i,j}(h)$, for $1 \le i \le n$, $-(n-1)/2 \le j \le (n-1)/2, 0 \le h \le (\ln V)/(lV)$, to be the set of points x in the unit square such that $(2i-2)l \le d_1(x) \le (2i-1)l$ and $(j - \frac{1}{2})h \le d_2(x) \le (j + \frac{1}{2})h$. Each $R_{i,j}(h)$ is a rectangle parallel to the diagonal with length l and height $h = O(l)$. We have $s \in R_{1,0}(h)$ and $t \in R_{n,0}(h)$.

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

We shall say that there is a path in the channel from s to t if there is a sequence of rectangles $R_{1,j_1}(h), R_{2,j_2}(h), \ldots, R_{n,j_n}(h)$ such that each rectangle contains at least one vertex and such that $j_1 = 0$ and $j_{k+1} = j_k \pm 1$, for $1 \le k < n$. This is illustrated in Figure 4. The path in the channel from s to t consists of $n-1$ edges with total length $\le \sqrt{2} + O(h^2/l^2)$. 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 $B(\sqrt{2} + O(h^2/l^2))$, which on the average is $O(hV/l)$.

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

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

By the product rule, this becomes

$$O\left(\frac{V}{l}\left(hP(h)\Big|_0^{(\ln V)/(lV)} - \int_0^{(\ln V)/(2lV)} P(h)\, dh\right)\right)$$

$$+ O\left(V\left(1 - P\left(\frac{\ln V}{lV}\right)\right)\right). \tag{2}$$

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

$$P(h) \ge 1 - 2 \sum_{1 \le k \le n/2} (k+1)k5^k q^{k+1}, \tag{3}$$

where $q < (1 - hl)^{V-2} < e^{-hl(V-2)}$ is the probability that a given rectangle $R_{i,j}(h)$ contains no vertices (other than possibly s or t). We derive (3) as follows: If there is no path from s to t in the channel, then there must be an "antipath" of empty rectangles that separate s from t. Let k, where $1 \le k \le n - 1$, be the number of nonempty rectangles in the longest contiguous chain of nonempty rectangles in the channel starting at s. Let us consider the case $k \le n/2$. There are k choices for the last nonempty rectangle in the chain. The antipath must contain at least $k + 1$ empty rectangles. At least one of the empty rectangles in the antipath must be adjacent to the last nonempty rectangle in the chain. The next k rectangles in the antipath can be chosen iteratively in $\le k5^k$ ways. The probability of a given set of $k + 1$ rectangles 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 rectangles in the channel contains k nonempty rectangles, for $1 \le k \le n/2$, is bounded by $(k + 1)k5^kq^{k+1}$. Summing up on k and by symmetry for the k in the range $n/2 < k \le n - 1$, we find that the probability of no path in the channel from s to t is $\le 2\Sigma_{1 \le k \le n/2}(k + 1)k5^kq^{k+1}$. This proves (3).

By (3) we have

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

and

$$\int_0^{(\ln V)/(lV)} P(h)\,dh \ge \int_{2/(lV)}^{(\ln V)/(lV)} P(h)\,dh$$

$$\ge \frac{\ln V}{lV} - \frac{2}{lV} - 2\int_{2/(lV)}^{(\ln V)/(lV)} \sum_{1 \le k \le n/2} (k + 1)k5^kq^{k+1}\,dh.$$

We can bound the integral by

$$\sum_{1 \le k \le n/2} (k + 1)k5^k \int_{2/(lV)}^{(\ln V)/(lV)} q^{k+1}\,dh$$

$$= \sum_{1 \le k \le n/2} (k + 1)k5^k \int_{2/(lV)}^{(\ln V)/(lV)} e^{-(k+1)hl(V-2)}\,dh$$

$$= \sum_{1 \le k \le n/2} (k + 1)k5^k \left(-\frac{e^{-(k+1)hl(V-2)}}{l(V-2)(k+1)} \right)\Bigg|_{2/(lV)}^{(\ln V)/(lV)}$$

$$\le \sum_{1 \le k \le n/2} (k + 1)k5^k \frac{e^{-2(k+1)(V-2)/V}}{l(V-2)(k+1)}$$

$$= O\left(\frac{1}{lV}\right).$$

Substituting our bounds into (2), we can bound the average spanning tree size by

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

The expected density is $\Theta(r^2)$. By (1), the average running time for the Euclidean heuristic is bounded by

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

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 us 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 $r(V) \geq 1$. Substituting this approximation into Lemma 1, we get an "approximate characterization" of the spanning tree at the end of the Euclidean heuristic as consisting of those vertices x such that

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

We can then apply (1) to bound the running time.

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 independently and 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 valid instances of the graph in the model. 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

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

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

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

By algebraic manipulation, we can show that each $x \in B$ satisfies $d_1(x) = O((r_2(V) - 1)/(r_1(V) - 1))$. For a constant fraction of points in this range, $d_2(x)$ can be $\Omega((r_2(V) - 1)/\sqrt{r_1(V) - 1})$. Hence, we have $|B| = \Omega((r_2(V) - 1)^2/(r_1(V) - 1)^{3/2})$. 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. Complete grids are worst-case graphs for the Euclidean heuristic, because all paths from s to t have the same length, namely, 2.

11. We assume that V is a perfect square. The V vertices are arranged in a $\sqrt{V}$ by $\sqrt{V}$ array. Each vertex can be connected only to its horizontal or vertical neighbors. (We refer to this as a "grid graph.")

If all $2\sqrt{V}(\sqrt{V} - 1)$ edges are present, then we say that the graph is a "complete grid." By Lemma 1, the spanning tree will consist of all V vertices, so the entire graph will be processed. If we break the grid restriction and 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 $\Theta(V)$.

In grid graphs it is more appropriate to use the l^1-metric (Manhattan distance) in place of *Eucl_dist*, since it provides a closer lower bound for *dist*. If we break ties in the priority queue by picking the vertex that was most recently modified among all those with the highest priority, then the heuristic will perform quite well.

The case of complete grid networks with diagonal edges, where $s = (0, 0)$ and t is an arbitrary gridpoint in the unit square, is studied in [Golden and Ball, 78]. The size of the spanning tree for the Euclidean heuristic is shown to be roughly an order of magnitude smaller than that for Dijkstra's algorithm.

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 average running time is $O(V)$, rather than $O(E + V \log V)$, 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.

The shortest paths problem can be studied 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 analysis for random graph models given in Theorem 1 holds if any lower bound for *dist*(x, t) is used, not just the Euclidean distance. Theorems 2 and 3 can also be generalized. For example, Theorem 2 also holds if we use the l^∞-metric (maximum of the scalar distances in each dimension) or the l^p-metric, for $p \geq 2$, to give a lower bound for *dist*(x, t). If the edges of the graph are constrained to be horizontal or vertical, then the l^1-metric (Manhattan distance) can be used to give a lower bound; the precise improvement for this heuristic has not been studied.

Our results hold even for entirely different metrics on $\Re^d$. Theorem 1 is true regardless of what metric is used to define the weights of the edges in the graph, as long as the same metric is used to compute the lower bound for *dist*(x, t) needed for the algorithm. Theorem 2 and a version of Theorem 3 are true for the l^p-metric, for $1 < p \leq \infty$, for example.

Another interesting problem deals with relaxing the requirement 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 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 complete 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 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.

The results of this paper suggest that it might be useful to study other problems dealing with Euclidean graphs. For example, it might be possible to compute a minimum spanning tree for a Euclidean graph in $O(E)$ time, on the average, assuming a random graph model in which the vertices are independently and uniformly distributed in the unit square. The best time bound for minimum spanning tree algorithms for general graphs is $O(E \log\log^* V)$, via a complicated algorithm in [Gabow, Galil, and Spencer, 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.

The analysis of Model 7 in Section 3 introduces an interesting theoretical question. By translation, we can consider channels as being paths in the two-dimensional directed lattice, restricted to the first quadrant. That is, each point (i, j), for $0 \leq i, j$, is connected via a directed edge to $(i + 1, j)$ and $(i, j + 1)$. We can assume that each lattice point is "passable" independently with probability p. The problem is to compute the probability that there is a directed path from $(0, 0)$ to $(1, 1)$ in the lattice involving only passable points. Inequality (3) gives a lower bound that is asymptotically equal to 1 when $p = 1 - q$ is very close to 1. The general question for other values of p (for example, constant values of p) is an interesting problem. It seems closely related to oriented percolation theory (see [Durrett, 84], for example), where similar questions are asked for the case in which the directed edges, not the points, are passable independently with probability p.

Acknowledgements. We would like to thank Bob Tarjan and Nick Pippenger for providing references to past work, and the referees for several helpful comments.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA (1974).
2. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerishe Mathematik*, 1 (1959).
3. R. Durrett. Oriented Percolation in Two Dimensions. *The Annals of Probability*, 12, 4 (December 1984), 999–1040.
4. 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.
5. 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), 338–346. The longer version of the paper will appear in *Journal of the ACM.*
6. H. N. Gabow, Z. Galil, and T. H. Spencer. Efficient Implementation of Graph Algorithms Using Contraction. *Proc. 25th Annual Symposium on Foundations of Computer Science*, West Palm Beach, FL (October 1984), 347–357.
7. B. L. Golden and M. Ball. Shortest Paths with Euclidean Distances: An Explanatory Model. *Networks*, 8 (1978), 297–314.
8. G. H. Gonnet. Expected Length of the Longest Probe Sequence in Hash Code Searching. *Journal of the ACM*, 28, 2 (April 1981), 289–304.
9. 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.
10. 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.

11. M. Loève. *Probability Theory*. Volume I. Graduate Texts in Mathematics, Springer-Verlag, New York (fourth edition 1977).
12. R. Sedgewick. *Algorithms*. Addison-Wesley, Reading, MA (1983).
13. R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA (1983).
14. J. S. Vitter. Faster Methods for Random Sampling. *Communications of the ACM*, 27, 7 (July 1984), 703–718.
15. A. C. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. *SIAM Journal on Computing*, 11, 4 (November 1982), 721–736.